PROYECTO FIN DE CARRERA
PLAN 2000

E.U.I.T. TELECOMUNICACIÓN

TEMA: "Wireless Sensor and Actuators Networks"

TÍTULO: Plataformas de desarrollo de WSAN y middleware para despliegue

AUTOR: Esther Moreno Sama

TUTOR: Vicente Hernández Díaz **Vº Bº.**

DEPARTAMENTO: DIATEL

Miembros del Tribunal Calificador:

PRESIDENTE: Matías Garrido González

VOCAL: Vicente Hernández Díaz

VOCAL SECRETARIO: Antonio Da Silva Fariña

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

Esta memoria tiene como principal objetivo ofrecer una solución middleware que implemente ciertas funcionalidades ofrecidas por la arquitectura middleware nSOM (nano Service-Oriented Middleware), desarrollado por la UPM, permitiendo su ampliación a otras plataformas hardware.

Se realizará un estudio del Estado del Arte actual y una comparativa de las diferentes plataformas hardware involucradas en las Redes Inalámbricas de Sensores y Actuadores (WSANs - Wireless Sensor-Actuator Networks). Este estudio tendrá como fin la elección de una de las plataformas para su futuro uso en un despliegue. Para dicha elección se prestará atención tanto a las tecnologías software como las tecnologías hardware, buscando retos tecnológicos.

Una vez elegida la plataforma se pretende contribuir en el middleware nSOM con la aportación de un despliegue parcial de dicho middleware en una WSAN con nodos con pocos recursos. Se conseguirá la publicación de las características y servicios ofrecidos por los nodos finales y el envío de peticiones y la recepción de respuestas.

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

Proyecto Fin de Carrera

Plataformas de Desarrollo de WSAN y Middleware para Despliegue de Aplicaciones

Autor
Esther Moreno Sama

Tutor
Vicente Hernández Díaz

Marzo 2014

A mis padres.

AGRADECIMIENTOS

A mis padres por haberme brindado la oportunidad de estudiar y formarme tanto en lo profesional como en lo personal. Agradecerles su paciencia y gran comprensión a lo largo de estos años. No ha sido un camino fácil pero siempre me habéis apoyado, haciendo que éste llegue a su fin. ¡Quien nos lo iba a decir! Muchas gracias.

A Laura que a pesar de resultarle difícil comprender el tiempo y sacrificio que supone esta carrera ha sido paciente todos estos años, siendo un importante e imprescindible pilar en mi vida.

A los que empezaron siendo compañeros de clase y ahora son buenos amigos, que me habéis acompañado en este viaje. No olvidaré aquellas tardes en los módulos y en la biblioteca. Sin vosotros esto no habría sido posible. Gracias.

Entre ellos quisiera hacer una mención especial a Álvaro, compañero y gran amigo desde que nos conocimos. Nunca podré agradecerte toda la ayuda que me has proporcionado, siendo mi principal apoyo dentro y fuera de la universidad. Siempre animándome de manera incansable e impidiendo que tirase la toalla. Muchas gracias.

Juntos en el cole, en la universidad y en la vida, mi amigo Luismi. Gracias por creer en mí y animarme en esos momentos que todo parecía perdido. Te espera un futuro muy prometedor y espero seguir siendo parte de él por mucho más tiempo. A Rocío, amiga también de la infancia, que en distintos momentos durante estos años me ha apoyado tanto en lo personal como en lo académico. A Sandra, amiga que formó parte de mis primeros años de universidad.

Agradecer a Vicente la confianza que ha depositado en mí y el esfuerzo que ha realizado a lo largo de este proyecto.

RESUMEN

Son muchos los dominios de aplicación que han surgido en los últimos años con motivo de los avances tecnológicos. Algunos como eHealth, Smart Building o Smart Grid están teniendo una gran aceptación por parte de empresas que incrementan sus inversiones en este tipo de campos. Las redes inalámbricas de sensores y actuadores juegan un papel fundamental en el desarrollo de este tipo de aplicaciones. A través de este tipo de redes inalámbricas es posible monitorizar y actuar sobre un entorno gracias a nodos sensores y actuadores de forma cómoda y sencilla.

Las WSANs (Wireless Sensors and Actuators Networks) junto con la robótica y M2M (Machine-to-Machine) están forjando el camino hacia el Internet of Things (IoT), un futuro en el que todo esté conectado entre sí. Cada vez aparecen dispositivos más pequeños y autónomos, que junto con el crecimiento de las redes, propician la interconexión de "el todo".

Este Proyecto Fin de Carrera tiene como objetivo contribuir en este avance, desarrollando parcialmente una solución middleware que abstraiga al usuario de la complejidad del hardware, implementando ciertas funcionalidades ofrecidas por el middleware nSOM desarrollado por la UPM.

Para conseguir este objetivo se realizará un estudio del Estado del Arte actual y una comparativa de las diferentes plataformas hardware involucradas en las Redes Inalámbricas de Sensores y Actuadores (Wireless Sensor-Actuator Networks). Este estudio tendrá como fin la elección de una de las plataformas hardware para su futuro uso en un despliegue parcial del mencionado middleware nSOM. Posteriormente, se diseñará e implementará un sistema para ejemplificar un caso de uso sobre dicha plataforma integrando la publicación de las características y servicios de cada nodo final y el envío de peticiones y la recepción de respuestas. Finalmente se obtendrá un conjunto de conclusiones a partir de los resultados obtenidos y se detallarán posibles líneas de trabajo.

ABSTRACT

There are many applications domains that have arisen because of technological advances in recent years. Some as eHealth, Smart Building or Smart Grid are having a great acceptance by companies that increase their investments in such fields. Wireless sensors and actuators networks play a fundamental role in the development of such applications. By means of this kind of wireless network it is possible to monitor and act upon an environment with the assistance of sensors and actuators nodes, readily.

The WSANs (Wireless Sensors and Actuators Networks) together with robotics and M2M (Machine-to-Machine) are forging the way towards the Internet of Things (IoT), a future in which all of them are connected among themselves. Smaller and more autonomous devices are appearing that, along with the growth of networks, foster the interconnection of 'the whole'.

This Degree Final Project aims to contribute to this breakthrough, developing partially a middleware solution that abstracts the user from the complexity of hardware, implementing certain functionalities offered by the nSOM middleware solution carried out by UPM.

To achieve this objective a study of the current state of the art and a comparison of the different hardware platforms involved in the Wireless and Actuators Sensor Networks (Wireless Sensor-Actuator Networks) will be performed. This study will aim the election of one of the hardware platforms for its future use in a partial deployment of the mentioned middleware nSOM. Subsequently, a system will be designed and implemented to exemplify a use case on the platform mentioned before integrating the publication of the features and services of each end node and sending requests and receiving responses. Finally a set of conclusions from the results will be stated and possible lines of future works will be detailed.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ACRÓNIMOS

6LoWPAN	IPv6 over Lowpower Wireless Personal Area Network
AC	Alternating Current
ADC	Analog-to-Digital Converter
AFH	Adaptive Frequency-Hopping
API	Application Programing Interface
ASCII	American Standard Code for Information Interchange
BLAST	Bursty, Light data, ASynchronous, Transitive
COM	Component Object Model
CSIRO	Commonwealth Scientific and Industrial Research Org.
CSMA/CA	Carrier Senser Multiple Access with Collision Avoidance
DAC	Digital-to-Analog Converter
DC	Direct Current
DH	Destination address High
DL	Destination address Low
DOS	Denial Of Service
EEPROM	Electrically Erasable Programmable Read-Only Memory
ETH	Eidgenössische Technische Hochschule
FCFS	First Come First Serve
FOS	Fleck Operating System
FPS	Fixed-Priority Scheduling
FTDI	Future Technology Devices International
GND	Ground
GNU GPL	GNU Lesser General Public License
GPS	Global Positioning System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HW	Hardware
I2C	Inter-Integrated Circuit
ID	Identifier
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers

IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific and Medical
ISO	International Standards Organization
JDK	Java Development Kit
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
LR-WPAN	Low Rate-WPAN
M2M	Machine-to-Machine
MAC	Media Access Control
MIT	Massachusetts Institute of Technology
NFC	Near Field Communication
nSOM	nano Service-Oriented Middleware
OEM	Original Equipment Manufacturer
OS	Operating System
OSI	Open System Interconnection
OTAP	Over The Air Provisioning
PAN	Personal Area Network
PC	Personal Computer
PDU	Protocol Data Unit
RAM	Random-Access Memory
RAR	Roshal ARchive
RF	Radio Frequency
RFID	Radio Frequency Identification
ROM	Read Only Memory
RTOS	Real Time Operating System
SCL	Simply Connect Last
SH	Serial number High
SICS	Swedish Institute of Computer Science
SL	Serial number Low
SM	Sleep Mode
SN	Number of Sleep periods

SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
SOS	Sensor network Operating System
SP	Sleep Period
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
ST	Time before Sleep
SW	Software
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UCLA	University of California
UML	Unified Modeling Language
UPM	Universidad Politécnica de Madrid
USB	Universal Serial Bus
UZ	Universidad de Zaragoza
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSAN	Wireless Sensors and Actuators Network
ZB	ZigBee

Capítulo 1:

Introducción

1. Introducción

Los continuos avances tecnológicos han permitido el desarrollo y crecimiento de las redes inalámbricas de sensores y actuadores (WSANs – Wireless Sensors and Actuators Networks). Son redes inalámbricas formadas por un grupo de nodos capaces de detectar y recoger características o cambios del entorno y actuar sobre el medio físico en el que se encuentran distribuidos.

A parte de los sensores y actuadores puede existir un tipo de nodo que combine ambas funcionalidades. Un ejemplo de esto, siendo una tendencia en las líneas de investigación, son los robots. Éstos estarán en movimiento, introduciendo un reto adicional que es introducirán la movilidad de todos o algunos de los nodos, captarán tanto las medidas de sus propios sensores como las de los nodos sensores cercanos en la WSAN y permitirán una reacción más eficaz ante estos datos. Estas características otorgan mejoras a las WSAN pudiendo responder dinámicamente y en tiempo real a los continuos cambios del entorno. La relación entre robótica y redes inalámbricas de sensores y actuadores les proporciona un mayor abanico de posibles aplicaciones, creando un sistema ubicuo que integre el mundo físico con el digital, descrito como Internet of Things (IoT).

El concepto IoT trata de la conexión de todo tipo de “cosas” en una red global que procese la información que generan elementos cotidianos y complejos. Esto ofrece la posibilidad de crear nuevos servicios y mejorar los ya existentes. Este concepto a su vez se ve estrechamente relacionado con el termino Machine-to-Machine (M2M), que hace referencia a las tecnologías que permiten el trafico bidireccional de información entre máquinas remotas de manera cableada o inalámbrica. Todos estos términos tienen en común la gran cantidad de datos que generan, de lo que surge el término Big Data, como el conjunto de tecnologías y técnicas que facilitan el procesado, análisis y almacenamiento de esta información.

A modo de contribución en la idea que engloban todos estos conceptos, la UPM ha desarrollado un middleware llamado nSOM, inspirado en “Service Oriented Architecture“(SOA) y usado en redes WSAN, para independizar a las aplicaciones basadas en dicha tecnología de los detalles y peculiaridades de las diferentes implementaciones de los diferentes proveedores. Este Proyecto Fin de Carrera pretende contribuir con la aportación de un despliegue parcial de dicho middleware en una WSAN con nodos con pocos recursos de computación, consiguiendo la publicación de las características y servicios ofrecidos por los nodos finales y el envío de peticiones y la recepción de respuestas.

1.1. Objetivos del Proyecto

Los objetivos de este Proyecto Final de Carrera son:

- El principal es **ofrecer una solución middleware que implemente ciertas funcionalidades ofrecidas por la arquitectura middleware nSOM (nano Service-Oriented Middleware)**. Este sistema desarrollado en este Proyecto Final de Carrera no tiene fines comerciales si no de investigación para el estudio de la ampliación del middleware nSOM a otras plataformas hardware.
- **Estudio actual y comparativo de las plataformas hardware recientes para WSANs (Wireless Sensors and Actuators Networks).** Primero se estudiarán tanto las tecnologías software como hardware de las redes inalámbricas de sensores y actuadores. Se realizará una tabla comparativa de plataformas hardware que permita el análisis de todas ellas en profundidad.
- **Elección de la plataforma sobre la que se va a ejemplificar un despliegue de middleware.** Este objetivo se llevará a cabo con la ayuda del estudio previamente hecho, buscando retos tecnológicos como consecuencia de las limitaciones hardware.
- **Desarrollo de una aplicación para el despliegue de un caso de uso.** Se diseñará y desarrollará un sistema que valide los resultados obtenidos en este proyecto y que sea capaz de integrarse en un futuro en la arquitectura de middleware nSOM. Para el diseño de este punto se tendrá en cuenta el paradigma SOA.

1.2. Motivos para la Realización del Proyecto

En los últimos años cada vez más y más las redes inalámbricas de sensores y actuadores atraen la atención de la comunidad científica, representando una de las tecnologías emergentes más importantes del siglo XXI. Además, conceptos estrechamente relacionados como IoT (Internet of Things), M2M (Machine-to-Machine), Robótica y Big Data, despertaron en mí un incipiente interés en poder trabajar y contribuir en este campo.

El proyecto planteaba una serie de retos tecnológicos que acogí con entusiasmo y valentía, el más significativo, trabajar con dispositivos con pocos recursos hardware y software.

El poder diseñar y desarrollar un sistema de estas características supone un gran avance en mi aprendizaje tanto académico como personal, afianzando conceptos ya adquiridos durante la carrera o nuevas disciplinas hasta ahora lejanas para mí.

1.3. Organización de Contenidos

- **Capítulo 1.** En este capítulo se presenta una introducción a este Proyecto Fin de Carrera. Se marcan los objetivos a lograr, como se han estructurado los contenidos y los motivos que llevan a la realización del proyecto
- **Capítulo 2.** Se presenta un estudio de Estado del Arte que pretende la familiarización con las Redes Inalámbricas de Sensores y Actuadores. Se estudiarán tanto las tecnologías software (sistemas operativos, lenguajes de programación, tecnologías inalámbricas, plataformas de desarrollo etc.) como las tecnologías hardware (arquitectura hardware de un nodo y plataformas hardware). Finalmente se presentará un estudio de mercado de las diferentes plataformas hardware disponibles en el mercado con el fin de elegir una que se adecue a los requisitos del proyecto.
- **Capítulo 3.** Desarrollo y despliegue del sistema realizado como caso de uso para la validación de parte de los resultados obtenidos en este Proyecto Final de Carrera. Se comenzará con una explicación y descripción global del diseño del sistema, continuando con la presentación de los casos de uso del sistema, los componentes, las clases y secuencia de la aplicación diseñada. También se plantearán los problemas encontrados a lo largo del proyecto. Por último se describirán las fases que se han de llevar a cabo para poner en funcionamiento el sistema diseñado. Además se facilitará un manual de usuario de la aplicación.
- **Capítulo 4.** Se analizarán los diferentes capítulos que conforman este Proyecto Fin de Carrera teniendo en cuenta los objetivos fijados. Esto permitirá extraer las conclusiones finales que ayudarán a valorar el trabajo realizado a lo largo de este proyecto.
- **Capítulo 5.** Se proponen posibles futuros trabajos que pueden suponer una mejora en el sistema desarrollado en este proyecto y la calidad de los servicios ofrecidos.

Capítulo 2:

Estudio del Estado del Arte

2. Estudio de Estado del Arte

La creación y evolución de las tecnologías están impulsando cada vez más a las Redes Inalámbricas de Sensores y Actuadores (WSAN), redes formadas por dispositivos de baja capacidad que pueden operar como nodos sensores o actuadores. Nuevos protocolos de comunicación (ZigBee, Bluetooth Low Energy, etc.) que ofrecen la posibilidad de conectar inalámbricamente dispositivos con un reducido consumo energético y sistemas operativos (TinyOS, ContikiOS) diseñados específicamente para WSAN, son algunos ejemplos de esta evolución. Además las aplicaciones y los beneficios que ellas conllevan potencian la investigación y desarrollo de todos los aspectos relacionados con las WSAN.

Muestra de esto, es que el Massachusetts Institute of Technology (MIT) publicó en 2003 un documento en el que clasificó a las Redes Inalámbricas de Sensores como una de las tecnologías emergentes más importante para el siglo XXI. Estas redes se definen como un conjunto de nodos distribuidos en un entorno para monitorizarlo y recolectar datos, procesarlos, transmitirlos a través de la red y, quizás, actuar sobre dicho entorno.

Uno de los factores más importantes a tener en cuenta cuando se pretende desplegar una red de este tipo es la plataforma hardware que va a formar parte de ésta. Desde el punto de vista del hardware (HW), tanto los elementos que se utilicen para el procesamiento, el tipo de sensores y actuadores, los dispositivos para comunicaciones o la fuente de energía pueden influir de forma crítica en el rendimiento de la plataforma y del sistema global. Y lógicamente los aspectos software (SW) como el sistema operativo y el lenguaje de programación que también son parte importante.

Por ello, a lo largo de este documento se estudiarán estos elementos de manera individual, analizando todas las ofertas disponibles en el mercado actual. Una vez concluido este estudio se llevará a cabo la elección de una plataforma hardware que mejor se adapte a las necesidades de un futuro despliegue.

2.1. Redes inalámbricas de Sensores y Actuadores

Una red inalámbrica de sensores y actuadores (del inglés Wireless Sensor and Actuator Network, WSAN) es una red inalámbrica ad-hoc formada por un conjunto de nodos autónomos e inteligentes (también conocidos como motas, por su ligereza y reducido tamaño), con capacidades de medida y recogida de datos (información del entorno como: temperatura, humedad, luz, movimiento…etc.) a través de los sensores que llevan incorporados, que se comunican entre sí o con una base central con el objetivo de intercambiar la información capturada por ellos mismos.

Los diferentes nodos pueden operar no solo como sensores, sino que sus capacidades de procesamiento y comunicación permiten que funcionen como actuadores, es decir, mecanismos capaces de activar procesos automatizados sobre sistemas externos bien siguiendo instrucciones remotas o a partir de decisiones tomadas de manera autónoma a partir de la información medida (por ejemplo, apertura de las ventanas de un edificio al detectarse un incendio, encender luces ante una disminución de la intensidad lumínica, etc.).

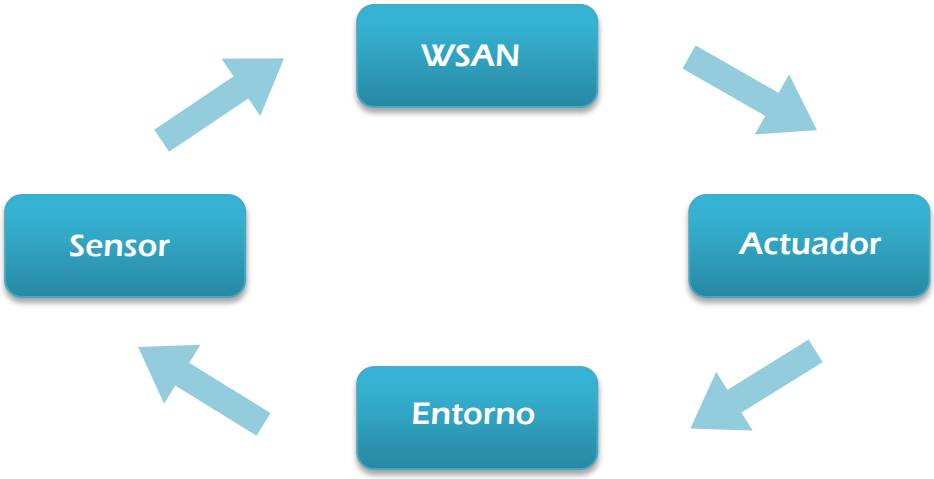


Figura 1. Redes Inalámbricas de Sensores y Actuadores.

Este tipo de redes tienen algunas características que no son propias de las redes inalámbricas de sensores:

- Los nodos actuadores, a diferencia de los nodos sensores, consumen más energía, son más caros y disponen de mayor capacidad de procesamiento y de comunicación.
- Es necesario que los actuadores reaccionen rápidamente ante los eventos, por lo tanto se deben introducir mecanismos de coordinación entre éstos y los sensores.
- En las WSAN suele haber un menor número de nodos actuadores frente a nodos sensores.

Los nodos se encuentran distribuidos geográficamente sobre el fenómeno de interés o muy cerca de él para monitorizarlo, tienen capacidad limitada de procesamiento y comunicación, y habitualmente un tiempo de vida limitado que depende de una batería

adjunta. Estas redes deben adaptarse a los cambios (posición, alimentación, etc.) puntuales en los diferentes nodos que las conforman.

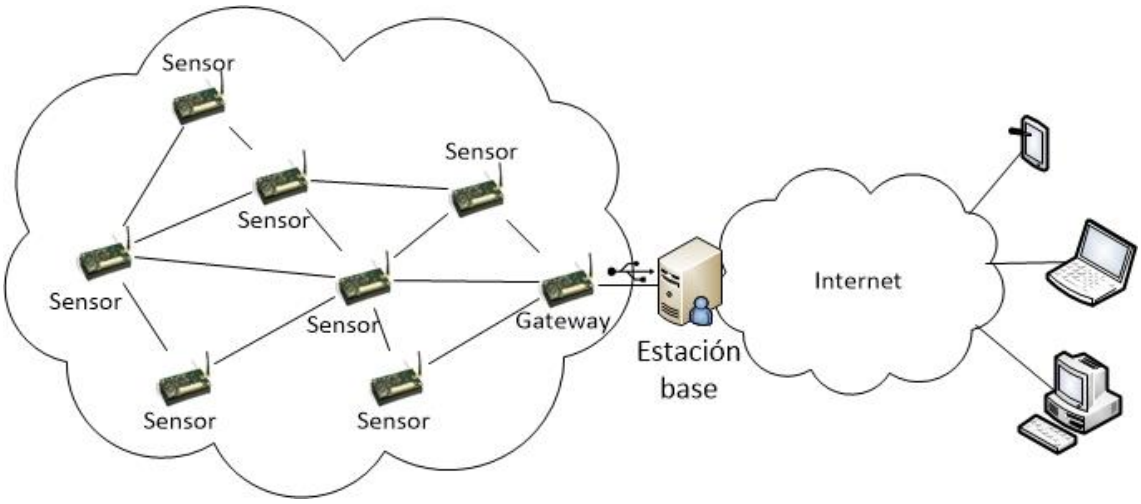


Figura 2. Arquitectura de una WSAN

Las redes inalámbricas de sensores se componen de cuatro elementos:

- **Nodos**: dispositivos electrónicos capaces de recoger la información recibida por los sensores, procesarla y transmitirla inalámbricamente a otros nodos o a la estación base. Para llevar a cabo dichas tareas es necesario:
  - ✓ Módulo de comunicaciones radio.
  - ✓ Microcontrolador.
- **Placas de sensores**: conjunto de sensores y actuadores de distinto tipo capaces de medir las condiciones y la información deseada del lugar en el que se encuentran. Su tamaño debe de ser reducido y tener un consumo bajo de energía. Se conectan al nodo a través del conector de expansión. A través de los sensores se pueden medir, entre otros muchos:
  - ✓ Variables medioambientales: luz, temperatura, humedad, presión, caudal, acústica, sísmica, barométrica etc.
  - ✓ Inercia: acelerómetro, inclinómetro, magnetómetro, giroscopio, etc.
  - ✓ Posicionamiento: GPS.
  - ✓ Químicos: CO2, CO.
- **Gateway**: elemento cuyo objetivo es actuar de puente para la interconexión entre dos redes de diferente tipo. Lo más habitual es que la red de sensores a una red de

datos (TCP/IP). Se conecta a la estación base por cable a través de USB, RS-232…y además tiene capacidad de comunicación inalámbrica. Posee dos funciones:

- ✓ Reprogramador de los nodos: descarga del código ejecutable en los microcontroladores.
- ✓ Pasarela entre la WSAN y la estación base (bidireccional).

- **Estación base**: elemento que se encarga de recolectar todos los datos provenientes de la red de sensores. Suele tratarse de un dispositivo de mayor capacidad para el almacenamiento, análisis y procesamiento de los datos procedentes de los nodos, como puede ser un ordenador común.

Las características principales de WSAN, que afectan directamente a la elección de una plataforma hardware y al diseño de la red, son las siguientes:

- **Tiempo de vida.** Una restricción que afecta a las WSAN es la energía, por lo que el funcionamiento de la red depende de la energía disponible en los nodos que la forman. Por lo tanto, es importante conocer qué elemento o elementos de un nodo consumen más energía, siendo éstos el microcontrolador y el módulo de radiocomunicación inalámbrica. Las principales fuentes de energía de las WSAN suelen ser las baterías, por lo que el tiempo de vida y el funcionamiento de la red dependen de cómo se aprovecha la energía. Es necesario que la red disponga de una vida útil del orden de uno o más años y para lograr este objetivo, es necesario que la red opere con un ciclo de trabajo lo suficientemente bajo. El ciclo de trabajo de una WSAN es el tiempo que está activa frente al tiempo que está inactiva o dormida. Por ello, los modos de gestión del consumo de los microcontroladores y de los módulos de radiocomunicación inalámbrica son decisivos y se deben explotar al máximo.
- **Coste reducido.** Es importante minimizar los cambios o recargas de las baterías, sobretodo en despliegues amplios o localizaciones hostiles, pues el coste es elevado en términos económicos. Además, como en la mayoría de las aplicaciones se pretende que los nodos no necesiten mantenimiento, explotando el concepto de nodos de "usar y tirar", se requiere que su coste sea mínimo y que realicen un aprovechamiento óptimo de la energía, maximizando el tiempo de vida útil, puesto que una vez desplegados no son recuperables.
- **Cobertura.** En algunas aplicaciones se requiere cubrir grandes áreas en espacios abiertos.
- **Interconectividad.** Las redes WSAN están planteadas para recoger información de una región concreta y que ésta pueda ser almacenada en una ubicación distinta en la que los recursos energéticos y de procesado no estén limitados. Para que pueda

llevarse a cabo, un punto clave es la interconexión entre la WSAN y otras redes o protocolos de transmisión que permitan enviar la información de la WSAN. Esta tarea es llevada a cabo por los sumideros, gateways, cuyos recursos son más adecuados para esta tarea de interconexión.

- **Heterogeneidad.** Las WSAN están compuestas por nodos con diferentes capacidades de cómputo y memoria, fabricante, funcionalidades, interfaces de comunicación… Se requieren nuevos algoritmos y protocolos de comunicación.
- **Privacidad y seguridad.** Puesto que las comunicaciones son inalámbricas la privacidad y seguridad son una preocupación, no obstante los mecanismos para la seguridad requieren recursos computacionales y energéticos que no siempre estarán disponibles.
- **Tiempo de respuesta.** Es importante acotar los tiempos de envío de información ya que este parámetro puede afectar al correcto funcionamiento de ciertas aplicaciones, como por ejemplo en aplicaciones de vigilancia y seguridad.
- **Recursos limitados.** Teniendo en cuenta que se tratan de sistemas empotrados, los recursos hardware son muy limitados. Es necesario, por lo tanto, minimizar el consumo de memoria (usando tipos de datos que consuman poca memoria) usando memoria dinámica (liberándola cuando sea posible) y controlando el tamaño del buffer de los puertos (evitando que se llenen y eliminándolos una vez usados). Además, se debe de programar de la manera más optimizada posible debido a que las APIs que se tengan que desarrollar tendrán restricciones de memoria y consumo energético.

2.2. Arquitectura Hardware

La arquitectura hardware de un nodo se puede dividir en varios bloques:

- **Bloque central:** proporciona la lógica computacional y de almacenamiento.
- **Bloque sensor/actuador:** define la funcionalidad del dispositivo, como sensor (medición de un parámetro) o como un actuador (capaz de interactuar con el entorno).
- **Bloque de alimentación:** proporciona energía al nodo.
- **Bloque inalámbrico de comunicaciones:** se encarga del intercambio de información entre nodos vecinos y/o gateway.

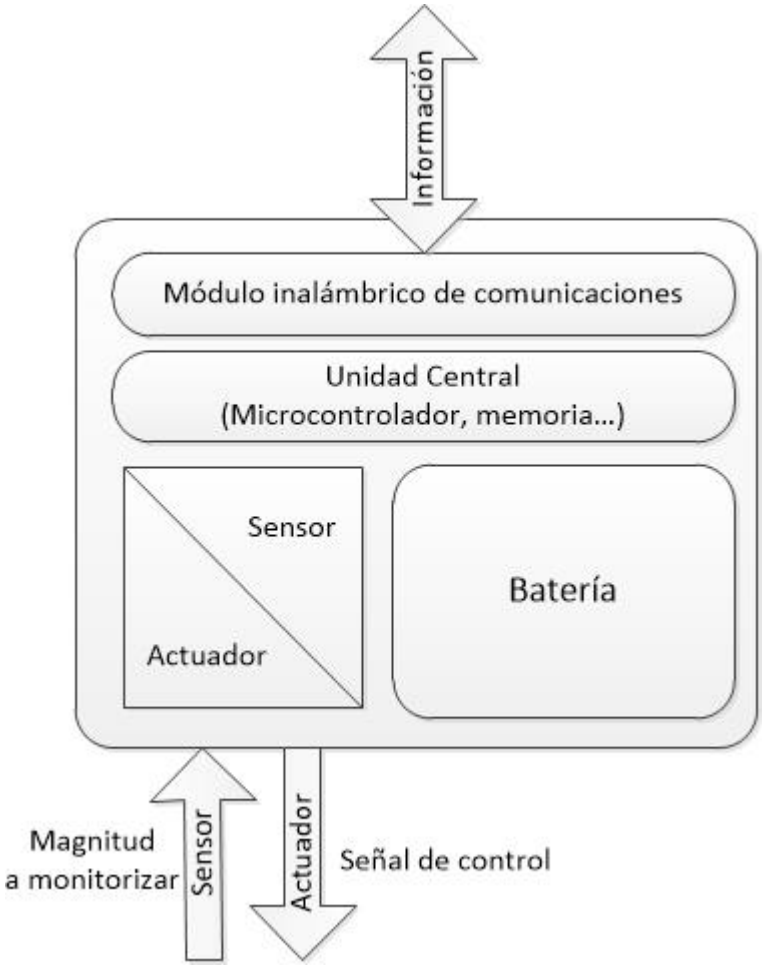


Figura 3. Arquitectura hardware de un nodo.

2.2.1. Microcontrolador

Se trata del elemento principal de la mota, ya que es el módulo encargado de procesar toda la información mediante tareas como la recolección de datos procedente de los sensores, la interpretación y procesado de esta información, la gestión de donde y cuando debe ser enviada dicha información, el control de los actuadores -si es que el nodo dispone de ellos- y en general cualquier operación que deba realizar el nodo.

También es el encargado de gestionar el almacenamiento de datos en la memoria, que en muchos casos, además de la memoria proporcionada por el microcontrolador, se incluye memoria externa adicional (por ejemplo, memoria flash). La memoria en un nodo sensor puede ser de tipo RAM, ROM o EEPROM.

El microcontrolador es un chip o pequeño circuito integrado compuesto de un microprocesador, memoria e interfaces para ADCs, DAC, UART, buses de interfaz serie

especializados como I2C, SPI, temporizadores y contadores. Los microcontroladores modernos frecuentemente incluyen un lenguaje de programación integrado.

Los microprocesadores usados en las redes inalámbricas de sensores presentan un consumo reducido, y en la mayoría de los casos disponen de varios modos de operación, siendo el consumo de energía en cada uno de estos diferente. Los más comunes son Active y Sleep. En este último modo el procesador solo mantiene alimentada la memoria y el tiempo de sincronización para realimentar (Wake-up) al procesador cuando sea necesario. Para alargar el tiempo de vida de las baterías es preferible que el procesador se encuentre la mayor parte del tiempo posible en estado Sleep. Por el contrario, en el modo Active -en el cual se realiza todo el proceso de adquisición y transmisión de datos- se debe estar el menor tiempo posible ya que es el estado donde se consume mayor cantidad de energía.

Hay muchos tipos de microcontroladores, actualmente los más utilizados son: ARM7, Cortex-M3, Atmel AVR, ATmega128x, TI MSP430, Intel 8051, etc. La mayoría de ellos tienen velocidades de reloj de unos pocos MHz.

2.2.2. Memoria

En general, los nodos no necesitan grandes cantidades de almacenamiento ni memoria para los programas. Los datos son almacenados temporalmente hasta que éstos son analizados y transmitidos a través de la red a la estación base o a un nodo vecino. Por regla general, los microcontroladores modernos incluyen un almacenamiento para programas entre 1kB y 128 kB, además pueden ser utilizados para almacenamiento temporal de datos. Adicionalmente contienen entre 32kB y 128kB de memora RAM que pueden ser utilizados para la ejecución de programas. Los requerimientos de memoria son muy dependientes de la capacidad que necesite la aplicación y de la topología de la red.

La memoria integrada en el microcontrolador y la memoria flash (externa) son las más relevantes desde el punto de vista energético. La memoria flash es una evolución de la memoria EEPROM que permiten la escritura y borrado de varias posiciones de memoria en una sola operación de programación mediante pulsos eléctricos. Son de carácter no volátil, de bajo coste y con gran capacidad de almacenamiento, características que hacen que sean utilizadas en redes inalámbricas de sensores.

2.2.3. Transceptor

El intercambio de datos entre los nodos es llevado a cabo por dispositivos de comunicación. Son varios los medios a elegir para realizar la comunicación:

radiofrecuencia, comunicación óptica, ultrasonido, satélite. Para redes inalámbricas de sensores la más adecuada es la radiofrecuencia.

Debido a que los nodos tienen limitada su capacidad de almacenamiento y que son desplegados típicamente en lugares de difícil acceso, se hace necesaria la presencia tanto de un transmisor como de un receptor en los nodos, siendo conveniente utilizar un dispositivo que combine ambas funcionalidades. Este tipo de dispositivos son llamados, transceptores. El transceptor permite la transmisión y recepción de información entre los diferentes nodos dentro de su rango de transmisión. Suele ser el mayor consumidor de energía, siendo los factores que influyen en este consumo el tipo de esquema de modulación utilizado, velocidad de datos, potencia del transmisor (determinado por la distancia de transmisión) y el ciclo de trabajo. Por ello, lo ideal en los dispositivos empotrados es permanecer el mayor tiempo posible en estado de bajo consumo o dormido, y despertarse para realizar operaciones o transmitir datos sólo cuando sea necesario.

En general los transceptores pueden operar en cuatro modos de operación: transmisión, recepción, *idle* (reposo) y modo *Sleep* (durmiendo). Una observación importante, en la mayoría de los transceptores, el modo de operación idle consume prácticamente lo mismo que el modo recepción, por lo tanto es mejor tener completamente apagadas las comunicaciones radio cuando no se está emitiendo ni recibiendo. También es un factor importante a considerar que el dispositivo presente un tiempo rápido de despertar (*wake-up*).

La gran mayoría de las plataformas hardware comerciales utilizan como protocolo de comunicación el estándar IEEE 802.15.4, diseñado específicamente para la monitorización y control remoto de aplicaciones. Éste define las características de las capas física y MAC para las redes inalámbricas de área personal de baja tasa de transmisión (LR-WPAN). Las limitaciones de memoria y procesado de los microcontroladores y módulos radio para LR-WPAN, llevaron a los vendedores a aceptar protocolos propietarios y soluciones para la conexión sobre 802.15.4 como Zigbee. La tecnología Zigbee es utilizada en un amplio abanico de productos y aplicaciones, define la seguridad y la capa de red y utiliza las capas ya definidas por el estándar IEEE 802.15.4.

2.2.4. Fuente de Energía

La fuente de alimentación es un componente crucial para la WSAN. En la mayor parte de los casos la fuente de energía de un nodo es una batería, aunque también pueden disponer de transformadores con salida adecuada para el nodo si se dispone de toma de corriente. Ésta suministra energía a todo el nodo, por lo tanto juega un papel vital en la determinación del tiempo de vida útil del nodo y del tamaño del nodo.

Las baterías son dispositivos complejos cuyo funcionamiento depende de muchos factores incluyendo el tamaño, tipo de material usado, etc. Para la gran mayoría de las plataformas comerciales o bajo desarrollo existentes, a menudo, se elige pilas de tipo AA, AAA y de botón, tales como las pilas alcalinas que ofrecen una densidad alta de energía a un precio muy bajo, contrarrestado por un gran tamaño físico respecto al nodo, una descarga no plana y una vida útil de sólo 5 años.

Puesto que la mayoría de las aplicaciones WSAN implican un despliegue de nodos de sensores en entornos alejados y hostiles se hace difícil el uso de métodos de recarga de baterías. Existen ciertas aplicaciones que permiten el uso de otras fuentes de energía externa, como es el caso de las técnicas de barrido y energías renovables, utilizando recursos como la luz solar y el viento, por ejemplo a través de paneles solares. Éstos pueden ser utilizados para alargar el tiempo de vida útil del nodo.

Los componentes que más cantidad de energía consumen de un nodo son el microcontrolador, el transceptor y sensores, siendo el momento de la transmisión de datos cuando mayor gasto energético se produce.

2.2.5. Sensores y Actuadores

Los nodos sensores y actuadores son los encargados en una WSAN de detectar fenómenos físicos y reaccionar ante ellos. Los nodos sensores son de bajo costo, bajo consumo energético, capacidad sensitiva limitada, bajo procesamiento y capacidad para comunicación a través de tecnologías inalámbricas. Los actuadores son nodos dotados con múltiples recursos con mejores capacidades de procesamiento, potencias de transmisión más elevadas y una batería con mayor vida útil.

Los sensores de las WSAN traducen los fenómenos físicos en señales eléctricas, y pueden clasificarse en dispositivos analógicos o digitales en función de la salida que producen. El consumo energético en los sensores viene dado principalmente por el muestreo de la señal y conversión de las señales físicas a eléctricas, acondicionamiento de la señal y la conversión de analógico a digital.

Existen muchos tipos de sensores para la monitorización de gran variedad de fenómenos físicos, desde sensores sencillos de luz y temperatura hasta complejos sistemas de sensores.

2.3. Sistemas Operativos

Es el encargado de gestionar los recursos hardware entre las distintas tareas que se ejecutan en el equipo, tales como el acceso a la memoria, tareas que se ejecutan en el procesador, etc. Los sistemas operativos utilizados en los PC difieren ampliamente de los desarrollados para las redes inalámbricas de sensores por dos motivos fundamentales:

- Existen importantes restricciones debido a los recursos hardware encontrados en los dispositivos. La capacidad de procesamiento y almacenamiento es limitada debido al tamaño de los dispositivos y a que están enfocados al bajo consumo.
- Las aplicaciones que se desarrollan sobre estos sistemas operativos tienen requisitos especiales. Por ejemplo, mientras que las aplicaciones para PC son interactivas y necesitan de interfaces de usuario que hagan sencilla su utilización, las aplicaciones para WSAN no lo necesitan.

A la hora de valorar y comparar los sistemas operativos de los dispositivos que forman parte de una WSAN hay que considerar el tiempo de respuesta y el nivel de multiprogramación:

- Sistema en tiempo real: se asegura a cada tarea que se ejecutará en un tiempo máximo. Además pueden existir distintas políticas, o prioridades, para gestionar el tiempo asignado de cada recurso.
- Sistema multitarea: a través de varios niveles de prioridades se distribuyen los recursos hardware entre las distintas tareas que se están ejecutando.

2.3.1. TinyOS

Se trata del primer sistema operativo desarrollado específicamente para WSAN, creado por la Universidad de Berkeley, programado en lenguaje nesC y multiplataforma. Es de código libre y está diseñado para cubrir las necesidades y características de las redes inalámbricas de sensores.

Para la planificación, se basa en una estructura de dos niveles: tareas y eventos. Los eventos tienen mayor prioridad que las tareas, permitiendo ejecutarlas y siendo interrumpidas esporádicamente por los eventos, que tardan menos en completar su ejecución. Además, estos eventos son gestionados mediante FCFS (First Come First Serve). Así, se pretende eliminar la necesidad de reservar espacio de memoria para guardar el contexto de los hilos en ejecución, que plantea el modelo basado en hilos. [**1**]

2.3.2. Contiki OS

Sistema operativo multiplataforma de código abierto desarrollado para sistemas hardware de baja capacidad, como sistemas empotrados, ordenadores de 8-bit y nodos de redes de sensores. Programado en lenguaje C. La configuración típica de Contiki ocupa tan solo 2kilobytes de RAM y 40 kilobytes de ROM. Fue creado por Swedish Institute of Computer Science (SICS). [**2**]

El núcleo está orientado en eventos y de manera opcional se puede planificar a través de hilos. Para que consuma menos recursos, realiza la carga dinámica de los procesos a través de una partición de la memoria del sistema en la compilación. Además los eventos generados por las tareas de Contiki pueden ser procesados de manera síncrona o asíncrona. Cuenta con implementaciones de una pila completa sobre protocolos y capas de adaptación definidos en 6LoWPAN.

2.3.3. FreeRTOS

Sistema operativo multiplataforma de código abierto y escrito en lenguaje C, está diseñado para sistemas en tiempo real. El núcleo de FreeRTOS puede a ocupar tan solo 4 kilobytes de memoria ROM y dependiendo de la arquitectura sobre la que se esté usando alcanzar los 9 kilobytes. [**3**]

No existe restricción en el número total de tareas que pueden ser ejecutadas ni en el número de prioridades, incluso varias tareas pueden tener la misma prioridad. Se utilizan distintos algoritmos para la planificación del procesador, pudiendo ser Round-Robin, sistemas de preferencia y modo cooperativo, siendo las propias tareas las que ceden el procesador a otras a través de la utilización de APIs. Éstas incluyen la sincronización y comunicación entre tareas, como semáforos, colas e indicadores de bloqueo.

2.3.4. Nut/OS

Se trata de un sistema operativo de código abierto que fue diseñado para ejecutarse en sistemas de BTnodes y es para aplicaciones en tiempo real. El núcleo está basado en hilos y utiliza mecanismos de sincronización y administración dinámica de memoria. Para manejo del hardware se proporcionan un conjunto de librerías y APIs que permiten la programación de aplicaciones. [**4**]

2.3.5. FOS (Fleck Operating System)

Sistema operativo creado para las motas Fleck de CSIRO. Proporciona un entorno de programación basado en hilos con soporte para dispositivos de entrada/salida, temporizadores… FOS proporciona un sistema basado en prioridades e hilos, no expulsivo con pilas separadas para cada hilo. Esto proporciona ventajas frente al modelo de programación concurrente, ya que no requiere semáforos. [**5**]

2.3.6. DuinOS

Sistema operativo basado en FreeRTOS para la plataforma hardware Arduino. Al igual que FreeRTOS, es un sistema en tiempo real con multitarea expulsiva. [**6**]

2.3.7. ArdOS

Sistema operativo diseñado especialmente para la plataforma hardware Arduino. El núcleo de ArdOS se basa en hilos, con multitarea y una planificación basada en prioridades para aplicaciones en tiempo real y planificación Round-Robin con quantos configurables para las aplicaciones de bajos requisitos de tiempo real. Para la coordinación de los hilos se utilizan: semáforos, mutex y variables de condición. Ocupa poca memoria, 1.6 kilobytes de memoria flash y menos de 200 bytes de RAM. [**7**] [**8**]

2.3.8. SOS

Es un sistema operativo desarrollado por la Universidad de California (UCLA), implementa un sistema que se comunica a través de mensajes y que permite múltiples hilos. Las aplicaciones son módulos que se pueden cargar o descargar en tiempo de ejecución, sin interrumpir el sistema operativo. Esto permite reconfigurar la red fácilmente, a través de actualizaciones de software, modificarlo, agregar nuevos nodos a la red e incluso quitar los módulos que se encuentren defectuosos. [**9**]

Actualmente, según la página oficial, no está siendo desarrollado. El sistema operativo aún está disponible para ser descargado, pero sus desarrolladores aconsejan utilizar otra de las múltiples alternativas que tengan soporte.

2.3.9. Nano-RK

Se trata de un sistema operativo desarrollado por la Universidad de Carnegie Mellon y que da soporte a redes multi-hops, siendo muy adecuado para las redes

inalámbricas de sensores. El núcleo del sistema operativo requiere poco espacio en memoria RAM, tan solo 2 kilobytes, y memoria ROM, 18 kilobytes. [**10**]

La planificación de tareas se realiza a través de multitareas expulsivas con prioridad, ya que se trata un sistema expulsivo basado en reserva bajo tiempo real. Las tareas con mayor prioridad pueden detener la ejecución de las que tienen menor prioridad, usa el algoritmo de planificación (FPS-Fixed-Priority Scheduling).

2.4. Plataformas de Desarrollo

Las plataformas de desarrollo son el conjunto de herramientas software que facilitan la programación software de aplicaciones para los diferentes sistemas operativos que se han mencionado en apartados anteriores. Ofrecen una mayor eficiencia a la hora de desarrollar programas.

Las plataformas de desarrollo suelen encontrarse ligadas a sistemas operativos concretos (Windows, Linux, Mac…), sobre los cuales se pueden ejecutar, y a los lenguajes de programación que se utilizarán para el desarrollo de las aplicaciones. En el caso de las redes inalámbricas de sensores, además es importante el sistema operativo de la plataforma hardware (TinyOS, ContikiOS, FreeRTOS…) sobre el cual se ejecutará la aplicación, puesto que la finalidad es crear aplicaciones para sistemas de bajo consumo de memoria y batería.

Es un aspecto importante a tener en cuenta puesto que sirve de gran ayuda a la hora de construir, integrar, extender, modernizar e implantar software en distintos sistemas.

2.4.1. Mote Runner

Se trata de la plataforma de desarrollo para redes inalámbricas de sensores creada por IBM [**11**]. Proporciona un kit de despliegue de software para facilitar la programación de aplicaciones en motas.

Mote Runner está diseñado para ejecutarse en controladores pequeños, estandarizados y embebidos incluyendo procesadores de bajo consumo de 8 bits, reduciendo tanto el coste inicial como los costes posteriores a la implementación y los costes de mantenimiento. Además ofrece a los programadores la posibilidad de utilizar lenguajes orientados a objetos como Java, permitiendo utilizar entornos como Eclipse, y C# para el desarrollo de aplicaciones WSAN que pueden ser distribuidas, cargadas, actualizadas y borradas dinámicamente incluso después de haber sido desplegadas. [**12**]

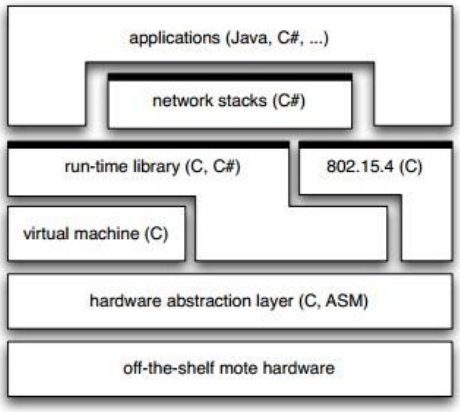


Figura 4. Arquitectura en capas de Mote Runner. IBM Mote Runner - Exectuve Summary

Esta plataforma soporta varios sistemas operativos, como Windows XP, MacOS X y Linux. Las motas Iris y Lotus del fabricante MEMSIC soportan esta plataforma de desarrollo.

2.4.2. MoteWorks

Es una plataforma de desarrollo para la creación de redes inalámbricas de sensores creada por MEMSIC. Proporciona un conjunto de herramientas software que apoya la creación eficaz y sencilla de soluciones WSAN. Está optimizada para redes de baja potencia y provee una solución extremo a extremo a través de todos los niveles de aplicación de una red inalámbrica de sensores y utiliza el lenguaje nesC para el desarrollo de aplicaciones. [**13**]

MoteWorks incluye TinyOS como sistema operativo además de los siguientes paquetes software:

- nesC compiler.
- Cygwin, entorno similar a Linux para sistemas operativos Windows.
- AVR Tools, conjunto de paquetes de desarrollo para procesadores Atmel AVR.
- Programmer's notepad, entorno de desarrollo integrado para la compilación y depuración de código.
- XSniffer, herramienta de monitorización de red para el entorno radio.
- MoteConfig, entorno gráfico para la programación OTAP (Over The Air Provisioning).
- Graphviz, permite visualizar archivos en forma de docs.

La plataforma MoteWorks está disponible para sistemas Windows (XP, 2000, NT, 7) y utiliza el sistema operativo TinyOS en las plataformas hardware.

2.4.3. IAR Embedded Workbench

Es una plataforma para el desarrollo de aplicaciones de redes inalámbricas de sensores. Incluye un sistema operativo RTOS, un middleware, un compilador y un depurador que permiten mejorar los diseños embebidos independientemente del microcontrolador escogido, puesto que soporta un amplio rango de componentes y plataformas hardware. Garantiza un bajo consumo de memoria y una gran portabilidad del código. La creación de aplicaciones se realizará a través de los lenguajes de programación C y C++. [**14**]

Soporta varias plataformas hardware como los microcontroladores AT86RF230, AT86RF231, AT86RF212 y ATmega128RFA1 del fabricante Atmel; MC1321x y MC1322x de Freescale; y C2430, CC2431, CC2530, CC2531 y CC2533 entre otros de Texas Instruments.

2.4.4. Eclipse y NESCDT

La plataforma de desarrollo Eclipse [**15**] incluye un conjunto de herramientas multiplataforma y de código abierto. Soporta un gran número de lenguajes de programación a través de una completa e intuitiva interfaz de usuario.

Para el desarrollo de aplicaciones para redes inalámbricas de sensores el equipo de TinyOS ha creado un plugin, llamado NESCDT [**16**], que permita programar de manera más cómoda y sencilla sobre dicho sistema operativo, utilizando como lenguaje de programación nesC. En este entorno de programación se encuentra además del editor un compilador y una herramienta para realizar depuraciones, que facilitan ampliamente la tarea de programación.

Al tratarse de un entorno multiplataforma es soportado por un gran número de sistemas operativos, entre ellos Windows, Linux y Mac.

2.4.5. Code Composer Studio

Es un entorno de desarrollo diseñado por Texas Instruments para las familias de procesadores embebidos. Proporciona un conjunto de herramientas para el desarrollo, compilado y depurado de aplicaciones embebidas. Incluye compiladores para cada familia

de dispositivos de Texas Instruments. Incluye además un editor, un entorno de creación de proyectos, un simulador y un sistema operativo en tiempo real. Está basado en Eclipse.

Code Composer Studio [14] está disponible para Windows y Linux (para algunas distribuciones, por lo que es necesario ver las características de la plataforma de desarrollo). Utiliza como lenguajes de programación C y C++.

2.4.6. Instant Contiki

Es un entorno de desarrollo para el sistema operativo Contiki. Contiene todas las herramientas necesarias para el desarrollo de aplicaciones sobre este sistema operativo, incluyendo compiladores, un depurador y un simulador. Está disponible para Ubuntu Linux y también para máquinas virtuales a través de VMWare Player. [**17**]

2.4.7. Waspmote IDE

Entorno de desarrollo para programar dispositivos Waspmote de Libelium, se incluyen las librerías necesarias para programar las aplicaciones. Se utiliza como lenguaje de desarrollo C++, lenguaje en el cual se encuentran desarrolladas también las librerías. Esta API ha sido desarrollada por Libelium para facilitar la programación de aplicaciones utilizando Waspmote y engloba todos los módulos que se integran en ésta, así como el manejo de otras funcionalidades como las interrupciones o los diferentes estados energéticos.

Incluye, junto con las librerías, un compilador y un entorno gráfico de desarrollo. Las motas Waspmote no disponen de sistema operativo. [**18**]

2.4.8. Arduino IDE

El IDE de Arduino [**19**] permitirá crear, abrir y modificar programas que definen lo que la placa va a hacer. El entorno de desarrollo de Arduino está escrito en Java por lo que es independiente del sistema operativo, pudiendo trabajar sobre cualquier sistema que soporte java como Windows, Mac OS X y Linux. Está basado en el lenguaje de programación Processing, avr-gcc y otros programas también de código abierto.

El entorno de desarrollo Arduino incluye un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes y una serie de menús. La placa de Arduino se conecta vía USB al ordenador y el IDE de Arduino permite cargar programas y comunicarse con ellos.

El IDE de Arduino permite escribir y editar código y convertir este código en instrucciones que el hardware Arduino entiende. Además transfiere las instrucciones de la placa Arduino (un proceso llamado *uploading*).

Este entorno de desarrollo trae una biblioteca en C++ llamada "Wiring" que hacen más fácil escribir los programas. El entorno de Arduino utiliza el concepto de un cuaderno de bocetos: un lugar estándar para almacenar los programas (o bocetos). El software escrito usando Arduino se llaman bocetos (en inglés, sketches).

2.5. Lenguajes de Programación

2.5.1. C

Es un lenguaje de programación de medio nivel con algunas características de bajo nivel. Está altamente extendido y se caracteriza por ser altamente portable, compacto y estructurado en módulos.

2.5.2. C++

Es un lenguaje hibrido y orientado a objetos. Es una extensión de C, pero con las modificaciones pertinentes para permitir el uso de objetos.

2.5.3. nesC

Se trata de un dialecto del lenguaje de programación C optimizado para las limitaciones de memoria de las Redes Inalámbricas de Sensores y orientado a componentes. [**20**]

2.5.4. C#

Lenguaje de programación orientado a objetos, desarrollado por Microsoft para formar parte de su plataforma .NET. Su sintaxis deriva de C/C++ y utiliza un modelo de objetos similar al de Java.

2.5.5. Java

Es un lenguaje de programación orientado a objetos, concurrente y basado en clases. Su sintaxis deriva de C y C++ aunque, a diferencia de estos, no proporciona facilidades de bajo nivel.

2.5.6. Arduino

El microcontrolador en la placa Arduino se programa mediante el lenguaje programación Arduino (basado en Wiring). Éste está basado en C y soporta todas las funciones del estándar C y algunas de C++. Los programas realizados con el lenguaje Arduino se denominan bocetos los cuales se dividen en tres partes principales: *estructura*, *valores* (variables y constantes), y *funciones. Éstos se crearán* utilizando el entorno de desarrollo (IDE) de Arduino, descrito en el Apartado 2.4.8, haciendo uso del editor de texto y se guardan con la extensión de archivo ".ino". [**21**] [**22**]

2.6. Tecnologías Inalámbricas

2.6.1. Estándar 802.15.4

La familia de estándares 802.15 especifica los requerimientos de las redes personales inalámbricas (WPAN, Wireless Personal Area Network). El propósito de 802.15.4 es definir estándares de red básicos para redes WPAN para dispositivos de bajo consumo y con bajas tasas de transmisión de datos.

El estándar 802.15.4 define las capas física y de gestión de acceso al medio. Esta última hace uso de CSMA/CA (Carrier Senser Multiple Access with Collision Avoidance) para permitir que varios dispositivos utilicen el mismo canal para el envío de información. [**23**]

Este estándar opera en las bandas 868MHz en Europa, 915MHz en EE.UU y 2,4GHz a nivel mundial, ofreciendo tasas de transferencia de entre 20kbps y 250kbps y rangos de cobertura de hasta 75 metros. [**24**]

2.6.2. ZigBee

Figura 5. ZigBee Alliance logo

ZigBee se trata de un conjunto de protocolos de alto nivel (abarca los niveles de red y aplicación) enfocados a la comunicación entre dispositivos de manera inalámbrica. El acceso al medio y el envío de mensajes se realiza utilizando el estándar 802.15.4. Esta especificación define una solución para comunicaciones inalámbricas que requieran un

bajo consumo, alta seguridad y con bajas tasas de tráfico. Éste conjunto de protocolos no está basado en IP (Internet Protocol). [**25**] [**26**]

La utilización del estándar 802.15.4 otorga a ZigBee las características de nivel físico y control de acceso al medio descritas anteriormente, tasas de tráfico de hasta 250kbps y cobertura de hasta 75 metros. Los valores pueden variar en función de las condiciones en las que se encuentre la red, viéndose alterada por interferencias, materiales de construcción de edificios, etc. Los rangos de cobertura de la red ZigBee desplegada pueden ser mucho mayores a los 75 metros gracias a los distintos tipos de topología que permite crear este conjunto de protocolos, entre ellas: topología en estrella, topología en árbol y topología mallada. [**26**]

Desde el punto de vista de las capas superiores de la arquitectura OSI, ZigBee define un conjunto de perfiles de aplicación de tipo público, creados por ZigBee Alliance, que están a disposición de todo el mundo. Además es posible definir perfiles privados creados por particulares o empresas para fines propios.

ZigBee puede incluir en una misma red hasta un máximo de 65535 nodos distribuidos en subredes de 255. A su vez, estos nodos se pueden clasificar según el cometido que vayan a cumplir: Coordinador Zigbee (controla y coordina la red), Encaminador (*router*) Zigbee (encargado de interconectar los dispositivos separados en la red) y Dispositivo Final, siendo su principal funcionalidad la transmisión de la información. [**23**]

Para ahorrar batería, los nodos ZigBee reducen el consumo permaneciendo en un estado "dormido", estarán a la espera de ser activado por un Router Zigbee o un Coordinador. Para comprobar que un Dispositivo Final continúa funcionando y no ha sido desconectado de la red se mandan mensajes periódicamente para despertar a este nodo.

2.6.3. DASH7

Figura 6. DASH7 logo

DASH7 es el nombre de la tecnología promovida por el consorcio sin ánimo de lucro DASH7 Alliance. Es un estándar open source para Redes Inalámbricas de Sensores, el cual opera en la banda del ISM sin licencia 433 MHz. DASH7 es para ISO 18000-7 lo que Wi-Fi es para IEEE 802.11. [**27**] [**28**]

La tecnología DASH7 aprovecha las ventajas básicas de la banda de 433 MHz para proporcionar una solución fiable y de baja complejidad a muchos problemas relacionados con la información. A esto lo llaman el concepto BLAST (Bursty, Light data, ASynchronous, Transitive).

El objetivo de DASH7 es ampliar el mercado de las tecnologías inalámbricas de bajo consumo mediante el aprovechamiento de la norma ISO 18000-7 ("DASH7" en si es un acrónimo suelto que significa Alianza de Desarrolladores para la Armonización de Estándares de ISO 18000-7). [**29**]

2.6.4. 6LoWPAN

6LoWPAN es un estándar del IETF. El primer documento del grupo de trabajo que describe el área del problema y el diseño de 6LoWPAN fue publicado en Agosto 2007. [**30**]

Es una tecnología de red que optimiza IPv6 para su uso con bajo consumo de energía y con tecnología de comunicación de bajo ancho de banda como IEEE 802.15.4. 6LoWPAN funciona comprimiendo 60 bytes de las cabeceras a solo 7 bytes y optimizando los mecanismos de trabajo en red en redes inalámbricas embebidas.

Uno de los principales inconvenientes de este protocolo es la seguridad, siendo vulnerable a simples ataques de DOS (Denial Of Service) con los cuales es posible "tirar" una red que haga uso de 6LoWPAN.

2.6.5. Bluetooth

Figura 7. Bluetooth logo

Se trata de una tecnología diseñada como solución de conectividad de corto alcance, de bajo consumo y bajo coste para dispositivos periféricos, portátiles u otros pequeños dispositivos electrónicos. Bluetooth [**31**] utiliza técnicas como Spread Spectrum, saltos en frecuencia (AFH) y señales full-duplex llegando a una tasa de 1600 saltos por segundo, dividida en 79 canales de 1MHz cada uno. El alcance es especifico de la clase de radio de la clase radio usada para la implementación: clase 1, alcance de 100 metros; clase 2, alcance de 10 metros; y clase 3, tiene un alcance de hasta 1 metro.

La tecnología con protocolo Bluetooth está basada en el estándar IEEE 802.15.1, utiliza la banda de frecuencia libre de 2,4 GHz, estableciendo la tasa máxima de transmisión en 3Mbps con un rango óptimo de alcance de 10 metros. Su uso está orientado para dispositivos de bajo consumo, con una cobertura baja y basada en receptores de bajo coste. [**24**]

La última versión de Bluetooth, llamada WiBree, se trata de una tecnología de radio digital diseñada para aplicaciones que requieran un consumo de energía ultra bajo y con un corto rango de alcance. Esta tecnología es compatible con Bluetooth y no pretende sustituirla si no complementarla.

2.6.6. Wi-Fi

Figura 8. WiFi logo

Wi-Fi es una tecnología inalámbrica de red que utiliza ondas de radio para proveer Internet inalámbrico de alta velocidad y conexiones de red. Un error común es creer que el termino Wi-Fi es una abreviatura de ”wireless fidelity”, sin embargo esto no es así. Wi-Fi es simplemente un término de marca registrada que significa IEEE 802.11x (a/b/g/n).

La Wi-Fi Alliance, la organización que posee el término Wi-Fi (marca registrada) define específicamente Wi-Fi como cualquier "producto WLAN (Wireless Local Area Network) basado en el estándar 802.11 del IEEE (Institute of Electrical and Electronics Engineers)”.

Wi-Fi en un principio se centró solo en el estándar 802.11b que trabaja en 2,4GHz, sin embargo Wi-Fi Alliance ha expandido el uso genérico del termino Wi-Fi para incluir todos los productos WLAN basados en el estándar 802.11, de esta manera se pretende evitar la problemática de la interoperabilidad entre WLANs.

2.7. Plataformas Hardware y Fabricantes

Existen numerosos grupos que han trabajado en el desarrollo de dispositivos destinados a las Redes Inalámbricas de Sensores y, en consecuencia, actualmente existen múltiples plataformas hardware disponibles en el mercado. En este apartado se muestra una selección de dichas plataformas, ya que sería imposible mostrar todas ellas, y una breve introducción sobre los fabricantes que disponen de esta tecnología. Algunas de las que se han sido excluidas debido a que son menos comunes en el mercado, son: CSIRO (Fleck3B), ETH Zurich (BTNode rev3), TinyNode (TinyNode 584), Advanticsys (CM3000), Inetsis (CoralMOTE IPv6), Texas Instrument (kMote), JENNIC (JN5248), Sun Microsystems (SunSPOT), etc.

2.7.1. Libelium

Figura 9. Libelium logo

La empresa Libelium [**32**] se creó en 2006 a partir de un trabajo realizado por dos estudiantes de la Universidad de Zaragoza (UZ), Alicia Asín y David Gascón, con el objetivo de convertirse en un proveedor de hardware horizontal y muy abierto en el mundo de M2M y las WSAN. En enero de 2007 consiguieron el estatus de Spin Off respecto de la UZ.

Se trata de una empresa especializada en el diseño y fabricación de hardware y tecnología para el desarrollo y la implementación de redes de sensores inalámbricas, redes malladas y protocolos de comunicación para todo tipo de redes inalámbricas distribuidas. Focalizando sus esfuerzos en el desarrollo de sensores y tecnologías de interconexión y desarrollo de aplicaciones en el contexto del Internet de las Cosas (IoT) y Machine-to-Machine (M2M).

La plataforma de Libelium está basada en su origen en recursos hardware como Arduino, siendo la primera aportación la mota SquidBee, pero en la actualidad ha derivado a sistemas propios creados por la propia compañía. Ésta ha desarrollado dos tipos de tecnologías: Waspmote que es la placa de sensores con los módulos de comunicaciones como Zigbee, 3G, WiFi o 802.15.4, y por otro lado el concentrador Meshlium, que recibe toda la información de los nodos y lo inyecta en las redes de comunicaciones. [**33**]

2.7.1.1. SquidBee

Figura 10. SquidBee logo

Al afrontar este proyecto se definieron los principales requisitos que debía tener SquidBee [**34**]. En cuanto a los sensores, se pensó que debería contar con un número variable que permitiese monitorizar diferentes espacios y situaciones, teniendo en cuanto esto se diseñó para permitir la conexión de hasta 18 sensores diferentes pudiendo ser añadidos de manera modular. Se conectan a la plataforma hardware como sondas de longitud variable, lo que permite medir, por ejemplo, humedad de la tierra en diferentes profundidades y del aire en dos alturas distintas con una única mota.

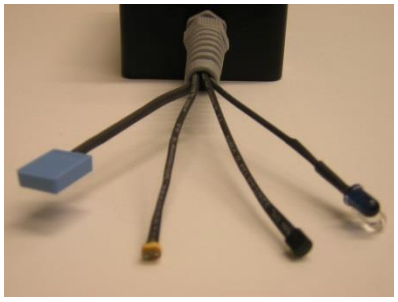
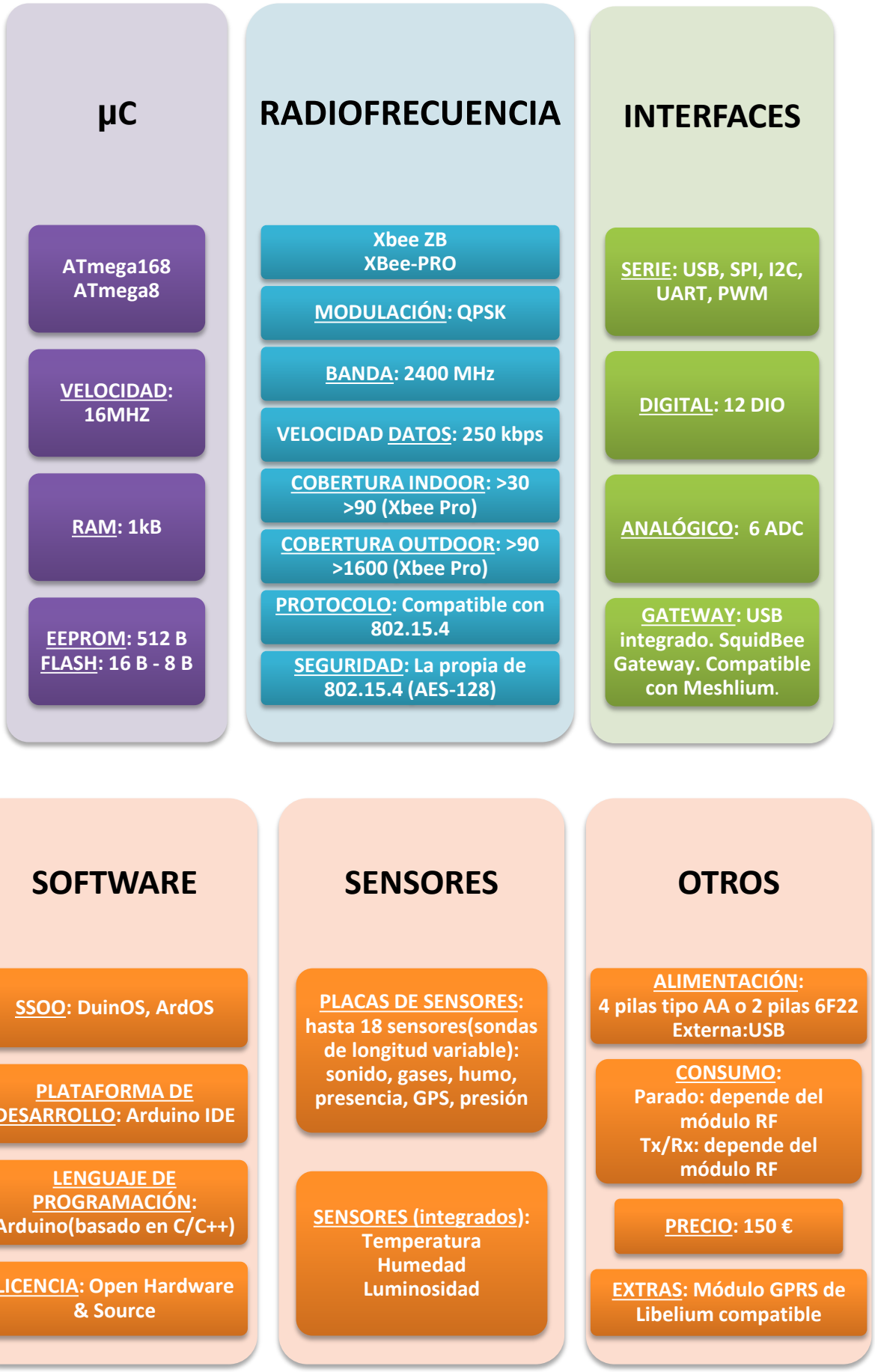


Figura 11. Detalle sondas de SquidBee: sensores de humedad, temperatura, luminosidad y presencia.

Por otro lado, era imprescindible elegir un modo de alimentación que permitiese a la mota ser completamente autónoma, siendo éste una pila de 9V que puede ser recargada por medio de una placa solar.

Figura 12. SquidBee.

Por último, la plataforma hardware debía ser capaz de transmitir los datos que recolectaba a distancias que estuviesen fuera del radio de alcance de la red sensorial. Para ello se equipó a la plataforma hardware con un módulo de comunicación ZigBee que permite crear redes punto a punto, punto a multipunto y malladas (mesh), ofreciendo un radio de transmisión de 1000 metros, que además puede incrementarse en varios kilómetros combinándolo con el router inalámbrico MeshLium. Éste actúa como pasarela entre los protocolos ZigBee y WiFi, consiguiendo una salida a Internet de los datos adquiridos. [**35**]


µC
ATmega168
ATmega8
VELOCIDAD: 16MHZ
RAM: 1kB
EEPROM: 512 B
FLASH: 16 B - 8 B
RADIOFRECUENCIA
Xbee ZB
XBee-PRO
MODULACIÓN: QPSK
BANDA: 2400 MHz
VELOCIDAD DATOS: 250 kbps
COBERTURA INDOOR: >30 >90 (Xbee Pro)
COBERTURA OUTDOOR: >90 >1600 (Xbee Pro)
PROTOCOLO: Compatible con 802.15.4
SEGURIDAD: La propia de 802.15.4 (AES-128)
INTERFACES
SERIE: USB, SPI, I2C, UART, PWM
DIGITAL: 12 DIO
ANALÓGICO: 6 ADC
GATEWAY: USB integrado. SquidBee Gateway. Compatible con Meshlium.
SOFTWARE
SSOO: DuinOS, ArdOS
PLATAFORMA DE DESARROLLO: Arduino IDE
LENGUAJE DE PROGRAMACIÓN: Arduino(basado en C/C++)
LICENCIA: Open Hardware & Source
SENSORES
PLACAS DE SENSORES: hasta 18 sensores(sondas de longitud variable): sonido, gases, humo, presencia, GPS, presión
SENSORES (integrados): Temperatura Humedad Luminosidad
OTROS
ALIMENTACIÓN: 4 pilas tipo AA o 2 pilas 6F22 Externa:USB
CONSUMO: Parado: depende del módulo RF Tx/Rx: depende del módulo RF
PRECIO: 150 €
EXTRAS: Módulo GPRS de Libelium compatible

2.7.1.2. Waspmote

Figura 13. Waspmote logo

Tras SquidBee se comenzó a trabajar en una nueva plataforma hardware que incorporará mejoras tanto a nivel software como hardware, siendo el principal objetivo a conseguir el ahorro de consumo energético. Las motas han sido programadas hasta el momento para transmitir datos cada cierto tiempo y permanecer en estado latente el resto del tiempo, el siguiente paso es conseguir que sea la propia mota quien decida cuándo despertarse en función de si hay o no necesidad de enviar información.

Figura 14. Waspmote.

En 2009 fue lanzada la primera versión de Waspmote (v1.1) [**36**] y desde entonces más de 2000 desarrolladores han estado usando la plataforma, enviando información de posibles mejoras. Libelium trabajó en estas mejoras tanto a nivel hardware como del API, teniendo como resultado el lanzamiento en 2013 de la segunda versión de la plataforma, Waspmote PRO (v1.2).

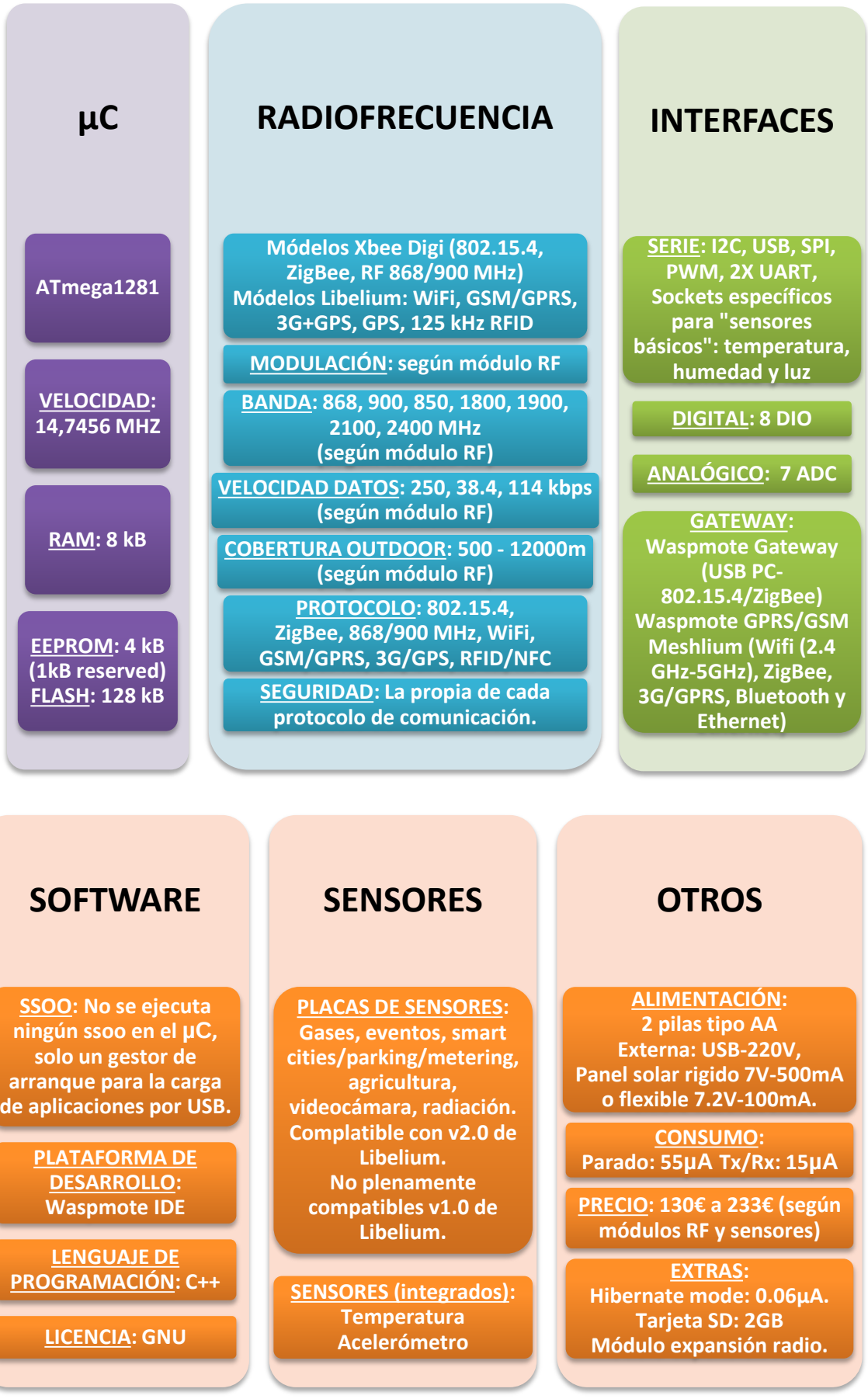
Waspmote [**37**] es un dispositivo sensorial orientado especialmente a desarrolladores basado en una arquitectura modular, es decir, integrar únicamente los

módulos que se necesiten en cada dispositivo y ser capaces de ampliarlos según las necesidades. Funciona con diferentes protocolos y frecuencias alcanzando distancias de hasta 12Km. Cabe destacar que en su modo de funcionamiento hibernate[1] de bajo consumo (0.06 µA) puede ahorrar el máximo de batería cuando no está transmitiendo permitiendo a los nodos vivir durante años ininterrumpidamente.

Con motivo de esta evolución [**38**] se han visto duplicados el número de sensores y protocolos de comunicación inalámbrica que soporta, incluyendo Wi-Fi, ZigBee, 802.15.4, Bluetooth, RFID/NFC, GSM/GPRS, 3G/GPRS, 868MHz, 900MHz e incluso la conmutación entre dos de estas tecnologías. Para ello se utilizará el módulo de expansión radio que permite conectar dos radios al mismo tiempo. En la Figura 15 se puede observar una Waspmote con estas características. Además se han creado 9 placas sensoriales (Gases, Eventos, Smart Cities, Smart Parking, Radiación, Video Cámara, etc.) que integran más de 50 sensores diferentes.

Figura 15. Waspmote con placa de expasión radio.

[1] El programa principal se detiene, el microcontrolador y todos los módulos de Waspmote quedan completamente desconectados. La única forma de volver a activar el dispositivo es a través de la alarma previamente programada a través del bus I2C en el RTC (interrupción síncrona). Al quedar el nodo totalmente desconectado de la batería principal el RTC es alimentado a través de una batería auxiliar de la que consume 0.06 µA.


µC
ATmega1281
VELOCIDAD: 14,7456 MHZ
RAM: 8 kB
EEPROM: 4 kB (1kB reserved) FLASH: 128 kB
RADIOFRECUENCIA
Módelos Xbee Digi (802.15.4, ZigBee, RF 868/900 MHz) Módelos Libelium: WiFi, GSM/GPRS, 3G+GPS, GPS, 125 kHz RFID
MODULACIÓN: según módulo RF
BANDA: 868, 900, 850, 1800, 1900, 2100, 2400 MHz (según módulo RF)
VELOCIDAD DATOS: 250, 38.4, 114 kbps (según módulo RF)
COBERTURA OUTDOOR: 500 - 12000m (según módulo RF)
PROTOCOLO: 802.15.4, ZigBee, 868/900 MHz, WiFi, GSM/GPRS, 3G/GPS, RFID/NFC
SEGURIDAD: La propia de cada protocolo de comunicación.
INTERFACES
SERIE: I2C, USB, SPI, PWM, 2X UART, Sockets específicos para "sensores básicos": temperatura, humedad y luz
DIGITAL: 8 DIO
ANALÓGICO: 7 ADC
GATEWAY: Waspmote Gateway (USB PC-802.15.4/ZigBee) Waspmote GPRS/GSM Meshlium (Wifi (2.4 GHz-5GHz), ZigBee, 3G/GPRS, Bluetooth y Ethernet)
SOFTWARE
SSOO: No se ejecuta ningún ssoo en el µC, solo un gestor de arranque para la carga de aplicaciones por USB.
PLATAFORMA DE DESARROLLO: Waspmote IDE
LENGUAJE DE PROGRAMACIÓN: C++
LICENCIA: GNU
SENSORES
PLACAS DE SENSORES: Gases, eventos, smart cities/parking/metering, agricultura, videocámara, radiación. Complatible con v2.0 de Libelium. No plenamente compatibles v1.0 de Libelium.
SENSORES (integrados): Temperatura Acelerómetro
OTROS
ALIMENTACIÓN: 2 pilas tipo AA Externa: USB-220V, Panel solar rigido 7V-500mA o flexible 7.2V-100mA.
CONSUMO: Parado: 55µA Tx/Rx: 15µA
PRECIO: 130€ a 233€ (según módulos RF y sensores)
EXTRAS: Hibernate mode: 0.06µA. Tarjeta SD: 2GB Módulo expansión radio.

2.7.2. MEMSIC

Figura 16. Crossbow y MEMSIC logos

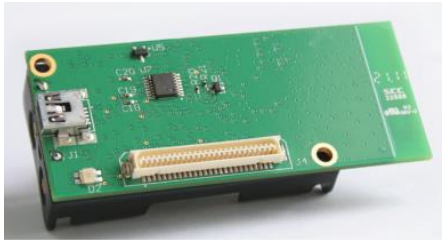
El grupo de investigación que desarrolló los primeros modelos de nodo para Redes Inalámbricas de Sensores pertenece a la Universidad de California, Berkeley. Los tres primeros modelos (Rene, Mica y Spec) fueron presentados por primera vez en la tesis doctoral de Jason Hill [**39**] desarrollados en esta universidad junto con el sistema operativo TinyOs. Crossbow [**40**], compañía norteamericana pionera en tecnologías de sensores, fue la primera en comercializar los modelos desarrollados por la Universidad de California. Fundada en 1995 fue líder en la provisión de soluciones extremo a extremo de redes inalámbricas de sensores y sistemas de sensores inerciales, que formaban parte de sus principales líneas de negocio.

Posteriormente surge MEMSIC [**41**] que adquiere la línea de productos de Crossbow relacionada con redes inalámbricas de sensores para dedicarse especialmente al desarrollo de plataformas hardware y software en este campo. [**42**]

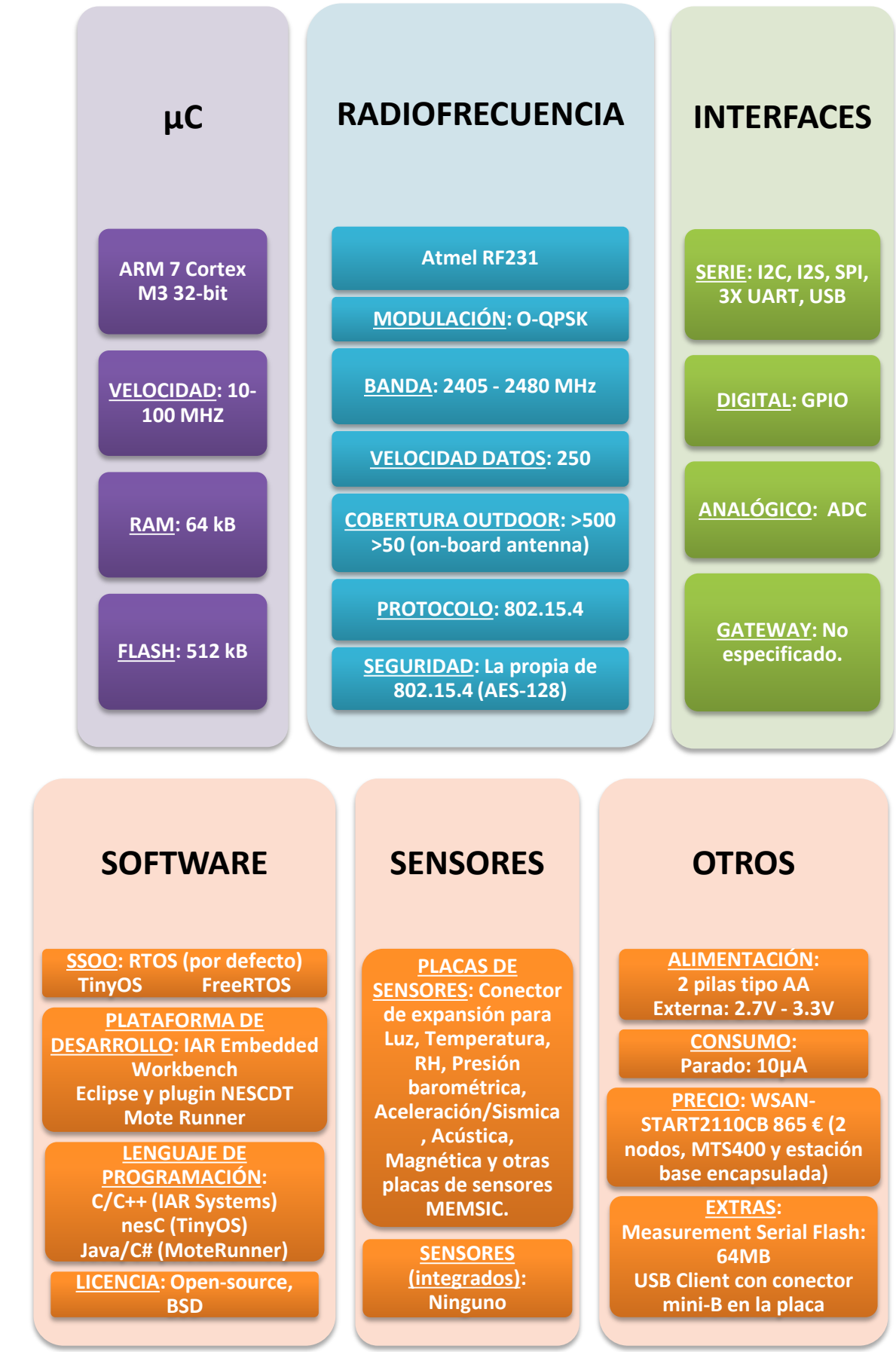
Una de las principales características de la arquitectura de todas las plataformas hardware (MICAz, MICA2, TelosB, IRIS, LOTUS) presentadas por Berkeley es la modularidad a nivel de sensores. La plataforma será siempre la misma a nivel de procesamiento o comunicaciones, pero en cuanto a los sensores pueden desarrollarse nuevas placas y a su vez nuevas aplicaciones. Todos estos nodos están basados en un µC y en el sistema operativo TinyOS.

2.7.2.1. Lotus

Lotus es la última generación de plataformas hardware para Redes Inalámbricas de Sensores de alto rendimiento presentada por MEMSIC [**43**]. Ofrece nuevas capacidades que mejoran la funcionalidad global del resto de productos y soluciones para Redes Inalámbricas de Sensores de MEMSIC.



Lotus es la evolución de I-Mote2, la cual es una versión comercial de Crossbow del diseño de Intel Mote 2, estando ésta bajo la licencia de Intel Corporation. Lotus [**44**] está desarrollada en torno al microcontrolador ARM7 Cortex M3 e incorpora lo mejor de IRIS, TelosB e I-Mote2, en una sola placa. Se basa en un diseño modular y apilable, que incorpora conectores para tarjetas de expansión. Es compatible con la gama de tarjetas de sensores y adquisición de datos MTS y MDA de MEMSIC, como la ITS400 igual a su antecesor I-Mote2. Lotus viene configurada de fábrica para ejecutar RTOS (Real Time Operating System), aunque están disponibles otras opciones como se verá en la siguiente tabla.


µC
ARM 7 Cortex M3 32-bit
VELOCIDAD: 10-100 MHZ
RAM: 64 kB
FLASH: 512 kB
RADIOFRECUENCIA
Atmel RF231
MODULACIÓN: O-QPSK
BANDA: 2405 - 2480 MHz
VELOCIDAD DATOS: 250
COBERTURA OUTDOOR: >500 >50 (on-board antenna)
PROTOCOLO: 802.15.4
SEGURIDAD: La propia de 802.15.4 (AES-128)
INTERFACES
SERIE: I2C, I2S, SPI, 3X UART, USB
DIGITAL: GPIO
ANALÓGICO: ADC
GATEWAY: No especificado.
SOFTWARE
SSOO: RTOS (por defecto) TinyOS FreeRTOS
PLATAFORMA DE DESARROLLO: IAR Embedded Workbench
Eclipse y plugin NESCDT
Mote Runner
LENGUAJE DE PROGRAMACIÓN:
C/C++ (IAR Systems)
nesC (TinyOS)
Java/C# (MoteRunner)
LICENCIA: Open-source, BSD
SENSORES
PLACAS DE SENSORES: Conector de expansión para Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.
SENSORES (integrados): Ninguno
OTROS
ALIMENTACIÓN:
2 pilas tipo AA
Externa: 2.7V - 3.3V
CONSUMO:
Parado: 10µA
PRECIO: WSAN-START2110CB 865 € (2 nodos, MTS400 y estación base encapsulada)
EXTRAS:
Measurement Serial Flash: 64MB
USB Client con conector mini-B en la placa

2.7.2.2. IRIS

Crossbow [**45**] anunció el lanzamiento de IRIS como una nueva familia de productos para Redes Inalámbricas de Sensores de ultra bajo consumo de energía y de largo alcance. En comparación con sus antecesores, como las motas MICA, tiene un radio de alcance tres veces superior y el doble de memoria de programa. Todo el software de aplicación y las placas de sensores de Memsic Inc. son compatibles con la mota IRIS.

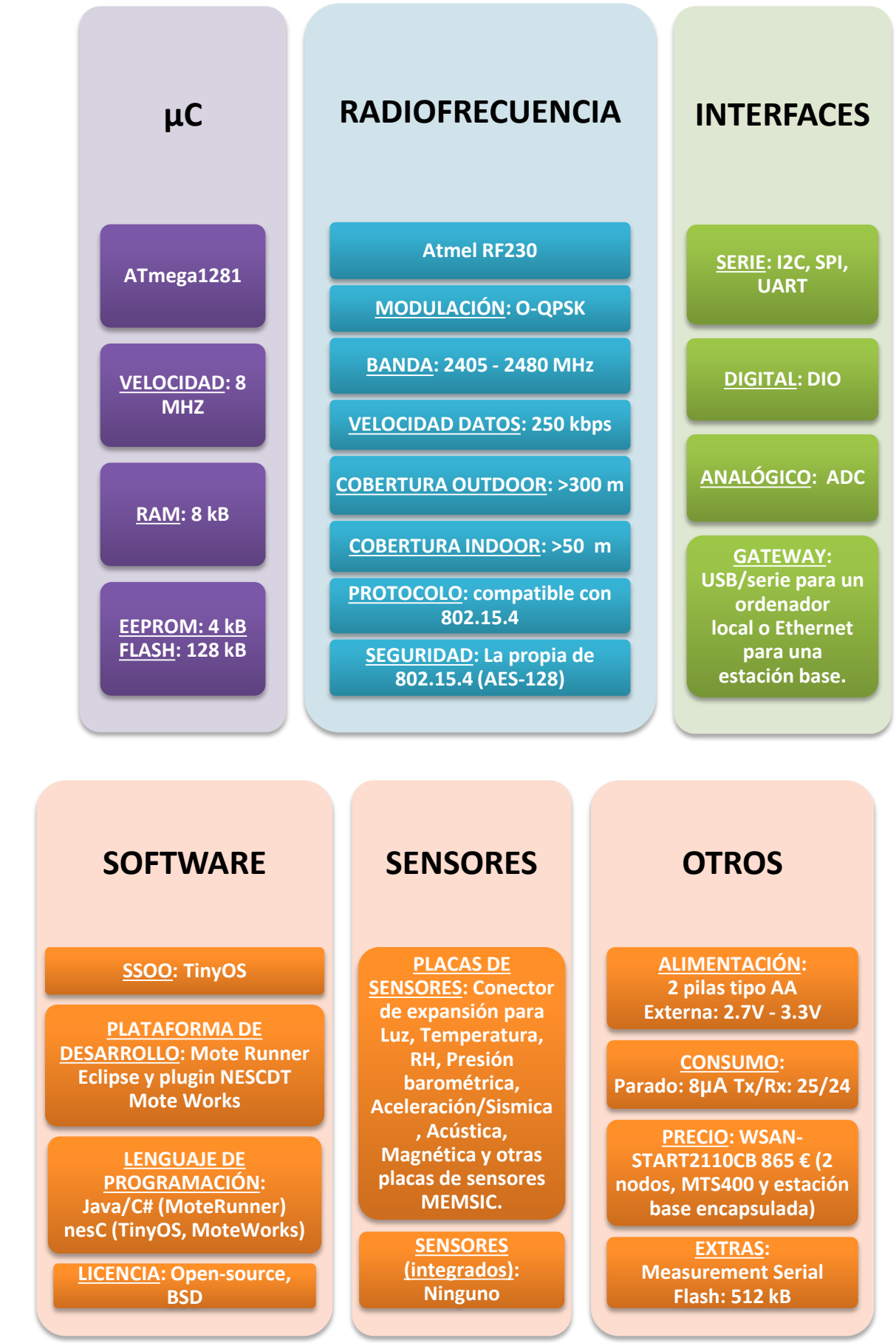
Figura 17. Mota IRIS

La nueva línea de productos IRIS que se presentó incluía un módulo Original Equipment Manufacturer (OEM) (M2110), una mota (XM2110), y un kit de diseño OEM, los cuales, actualmente siguen ofreciéndose por Memsic Inc. [**45**]

Figura 18. IRIS OEM EDITION

La mota IRIS puede funcionar como una estación base cuando está conectada a una interfaz PC o a un gateway.

En 2010 IBM anunció un contrato con Memsic Inc. en el cual se unían las motas IRIS con una nueva plataforma de software llamada Mote Runner, descrita en secciones anteriores, inventada por los científicos de IBM Zurich. Esta combinación de hardware y software ofrece a los clientes una plataforma fácil de usar para pruebas, depuración y mantenimiento de aplicaciones. [**46**]


µC
ATmega1281
VELOCIDAD: 8 MHZ
RAM: 8 kB
EEPROM: 4 kB
FLASH: 128 kB
RADIOFRECUENCIA
Atmel RF230
MODULACIÓN: O-QPSK
BANDA: 2405 - 2480 MHz
VELOCIDAD DATOS: 250 kbps
COBERTURA OUTDOOR: >300 m
COBERTURA INDOOR: >50 m
PROTOCOLO: compatible con 802.15.4
SEGURIDAD: La propia de 802.15.4 (AES-128)
INTERFACES
SERIE: I2C, SPI, UART
DIGITAL: DIO
ANALÓGICO: ADC
GATEWAY: USB/serie para un ordenador local o Ethernet para una estación base.
SOFTWARE
SSOO: TinyOS
PLATAFORMA DE DESARROLLO: Mote Runner Eclipse y plugin NESCDT Mote Works
LENGUAJE DE PROGRAMACIÓN: Java/C# (MoteRunner) nesC (TinyOS, MoteWorks)
LICENCIA: Open-source, BSD
SENSORES
PLACAS DE SENSORES: Conector de expansión para Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.
SENSORES (integrados): Ninguno
OTROS
ALIMENTACIÓN: 2 pilas tipo AA Externa: 2.7V - 3.3V
CONSUMO: Parado: 8µA Tx/Rx: 25/24
PRECIO: WSAN-START2110CB 865 € (2 nodos, MTS400 y estación base encapsulada)
EXTRAS: Measurement Serial Flash: 512 kB

2.7.2.3. MICAz

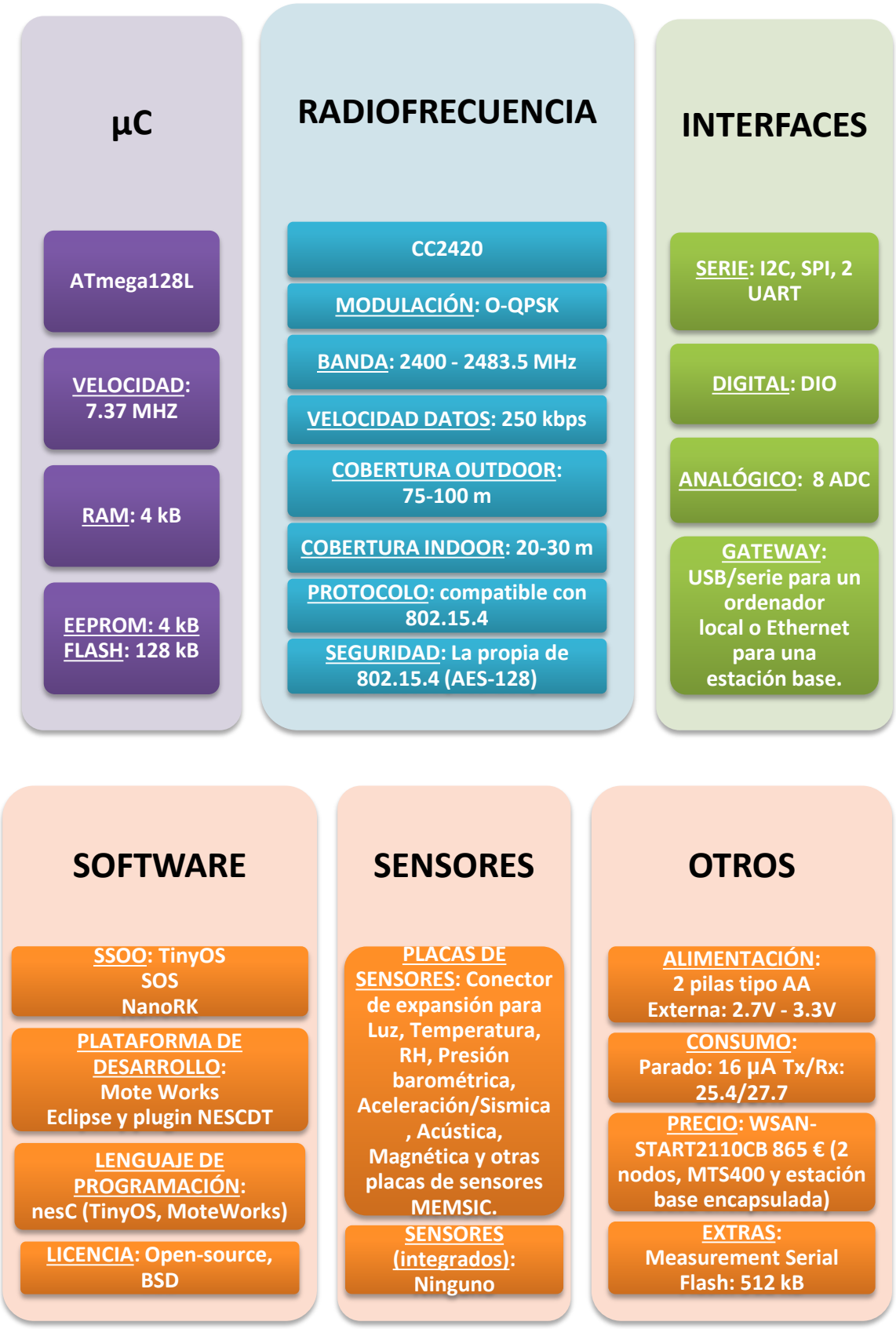
Los investigadores de la Universidad de California, Berkeley desarrollaron una plataforma open hardware y open source que combinaba sensores, comunicaciones y procesado en una única arquitectura, teniendo como resultado la primera generación comercial de esta mota, llamada Rene. Tras ésta, y con el fin de implementar mejoras, la U.C. Berkeley desarrolló la segunda generación de motas comerciales, llamada MICA. Posteriormente los investigadores completaron el diseño básico del hardware y Crossbow construyó varios miles de unidades, distribuyéndolas a los desarrolladores. Ésta primera mota MICA operaba en la banda de frecuencias de 916 o 433MHz y tenía 128kbytes de memoria flash. [**47**]

La tercera generación comercial de motas MICA, aumentaron la memoria flash a los 512kbytes. Son la mota MICA2, que sigue operando en las frecuencias 433MHz y 868/916MHz, y la MICAz, que es similar a la MICA2 pero la radio opera en los 2.4GHz, teniendo como resultado una velocidad de datos máxima de 250 Kbps. Éstas dos últimas actualmente son distribuidas por Memsic Inc. [**48**]

Figura 19. MICAz

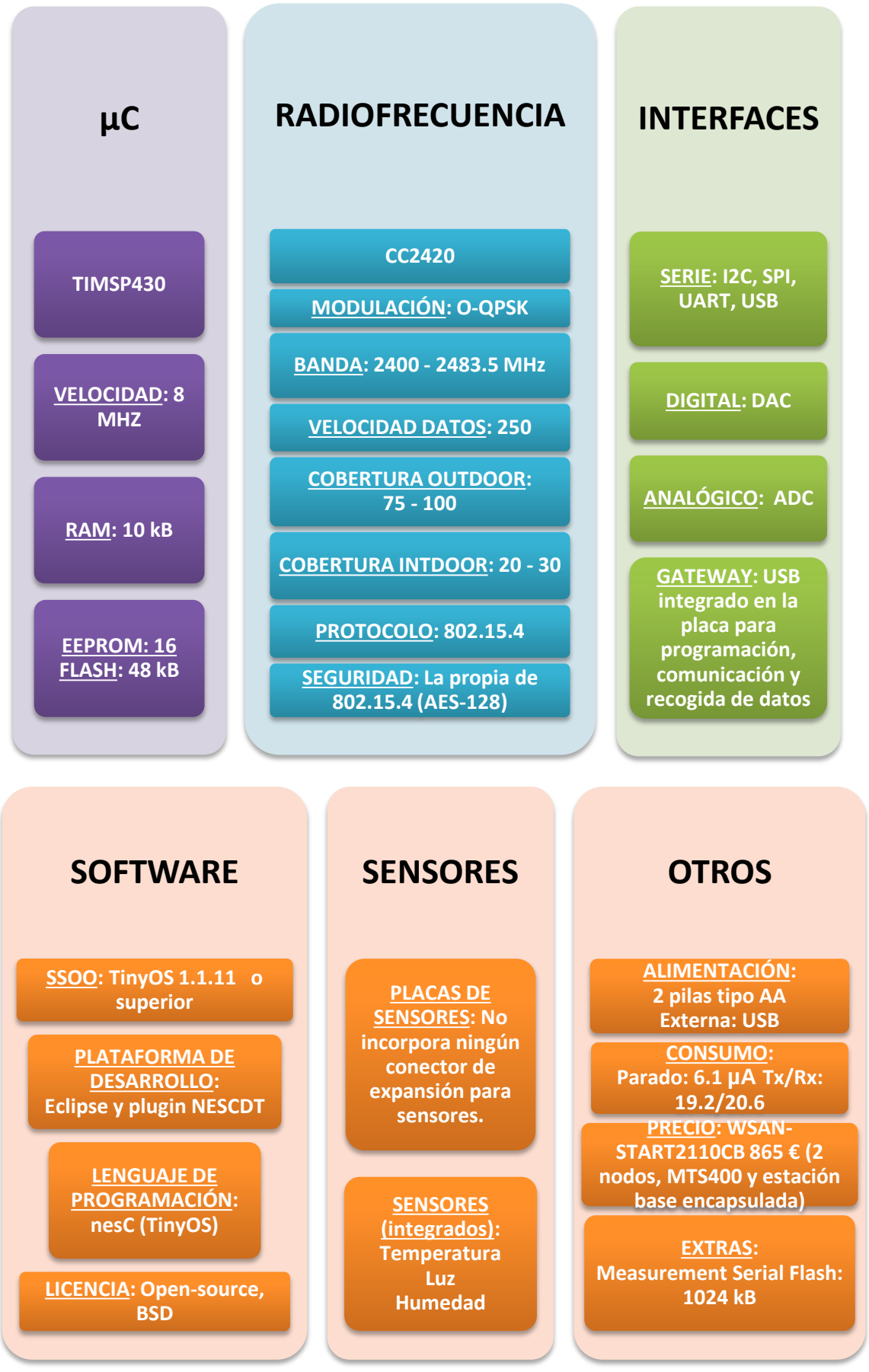
El diseño de la mota MICA consiste en una pequeña radio de bajo consumo y una placa para el procesador (llamada mota procesador/radio, MPR) y una o más placas de sensores (conocida como mota sensor, MTS). Los módulos MPR contienen diversas interfaces de sensores, que están disponibles a través de un conector de expansión de 51 pines que une los módulos MPR y MTS. [**49**]

A continuación se mostrarán las características técnicas de la mota MICAz:


µC
ATmega128L
VELOCIDAD: 7.37 MHZ
RAM: 4 kB
EEPROM: 4 kB
FLASH: 128 kB
RADIOFRECUENCIA
CC2420
MODULACIÓN: O-QPSK
BANDA: 2400 - 2483.5 MHz
VELOCIDAD DATOS: 250 kbps
COBERTURA OUTDOOR: 75-100 m
COBERTURA INDOOR: 20-30 m
PROTOCOLO: compatible con 802.15.4
SEGURIDAD: La propia de 802.15.4 (AES-128)
INTERFACES
SERIE: I2C, SPI, 2 UART
DIGITAL: DIO
ANALÓGICO: 8 ADC
GATEWAY: USB/serie para un ordenador local o Ethernet para una estación base.
SOFTWARE
SSOO: TinyOS SOS NanoRK
PLATAFORMA DE DESARROLLO: Mote Works Eclipse y plugin NESCDT
LENGUAJE DE PROGRAMACIÓN: nesC (TinyOS, MoteWorks)
LICENCIA: Open-source, BSD
SENSORES
PLACAS DE SENSORES: Conector de expansión para Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.
SENSORES (integrados): Ninguno
OTROS
ALIMENTACIÓN: 2 pilas tipo AA Externa: 2.7V - 3.3V
CONSUMO: Parado: 16 µA Tx/Rx: 25.4/27.7
PRECIO: WSAN-START2110CB 865 € (2 nodos, MTS400 y estación base encapsulada)
EXTRAS: Measurement Serial Flash: 512 kB

Con posterioridad a la generación de las motas MICA, aparecieron en el mercado las motas de la familia Telos, también desarrollados en la U. C. Berkeley y comercializados por distintas empresas bajo dos marcas: TelosB (Crossbow, 2005) y Tmote Sky (Moteiv).

TelosB y Tmote Sky, que si bien fueron fabricadas y comercializadas por empresas diferentes, Crossbow Technology y Moteiv respectivamente, son equivalentes. La producción de la mota Tmote Sky finalizó después de que Moteiv pasase a ser Sentilla Corp [**50**] [**51**]. Debido a este cese, solo se comentarán los aspectos técnicos de la mota TelosB.


µC
TIMSP430
VELOCIDAD: 8 MHZ
RAM: 10 kB
EEPROM: 16
FLASH: 48 kB
RADIOFRECUENCIA
CC2420
MODULACIÓN: O-QPSK
BANDA: 2400 - 2483.5 MHz
VELOCIDAD DATOS: 250
COBERTURA OUTDOOR: 75 - 100
COBERTURA INTDOOR: 20 - 30
PROTOCOLO: 802.15.4
SEGURIDAD: La propia de 802.15.4 (AES-128)
INTERFACES
SERIE: I2C, SPI, UART, USB
DIGITAL: DAC
ANALÓGICO: ADC
GATEWAY: USB integrado en la placa para programación, comunicación y recogida de datos
SOFTWARE
SSOO: TinyOS 1.1.11 o superior
PLATAFORMA DE DESARROLLO: Eclipse y plugin NESCDT
LENGUAJE DE PROGRAMACIÓN: nesC (TinyOS)
LICENCIA: Open-source, BSD
SENSORES
PLACAS DE SENSORES: No incorpora ningún conector de expansión para sensores.
SENSORES (integrados): Temperatura Luz Humedad
OTROS
ALIMENTACIÓN: 2 pilas tipo AA Externa: USB
CONSUMO: Parado: 6.1 µA Tx/Rx: 19.2/20.6
PRECIO: WSAN-START2110CB 865 € (2 nodos, MTS400 y estación base encapsulada)
EXTRAS: Measurement Serial Flash: 1024 kB

2.7.3. Arduino

Figura 20. Arduino logo

La mayoría de las placas Arduino son fabricadas por Smart Projects en Italia. Sin embargo, Arduino Pro, Pro Mini y LyliPad son fabricadas por SparkFun Electronics, en EE.UU. Por otro lado Arduino Nano es fabricado en Gravitech también situada en EE.UU. También es posible hacerlas a mano utilizando los diseños de referencia de la placa Arduino que están disponibles en su página web, esto es posible gracias a que es open hardware.

Arduino es una plataforma open source para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y también podrá intervenir en éste controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). [**21**]

En la página de oficial de Arduino se encontrarán diferentes versiones de placas Arduino, aquí se presenta una pequeña muestra de éstas:

Figura 21. Arduino Micro

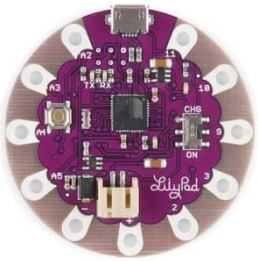


Figura 22. LilyPad Arduino USB

Figura 23. Arduino Nano

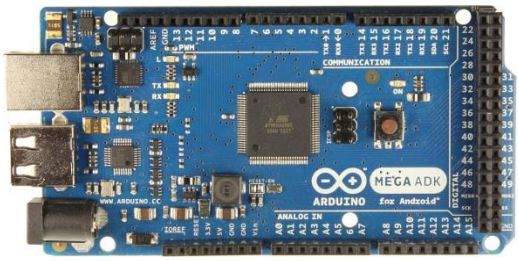


Figura 24. Arduino Mega ADK

Figura 25. Arduino Robot

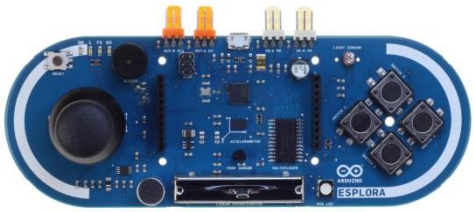


Figura 26. Arduino Esplora

2.7.3.1. ArduinoUNO (rev3)

ArduinoUNO [**22**] [**52**] es una placa basada en el ATmega328 y se trata de la versión optimizada de su predecesor Duemilanova. Incorpora una serie de mejoras:

- Nuevo bootloader OptiBoot que permite cargar programas a 115Kbps (56Kbps en la versión anterior), éste también ha sido reducido en tamaño pasando a ocupar sólo 512bytes dejando disponible más memoria flash para programas.

- Incorpora un ATmega16U2 programado como convertidor serie a USB, dejando de utilizar el chip convertidor serie a USB FTDI de versiones anteriores. El nuevo chip permite ser reprogramado para que la placa sea reconocida, como cualquier tipo de periférico USB, al conectarla al ordenador.

- Se han añadido más pines: SDA y SCL para comunicación I2C y en el bloque de alimentación se incorporan dos pines, IOref que servirá para que la placa reconozca

el tipo de alimentación que requieren los shields (3.3V o 5V) y al otro se le dará uso en futuras versiones.

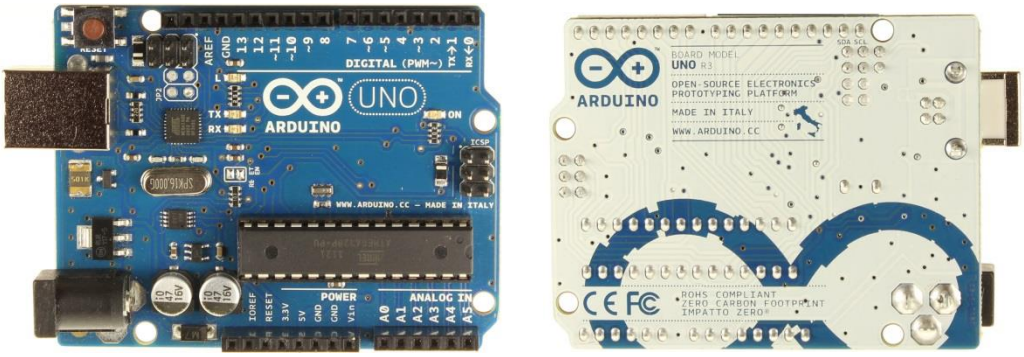
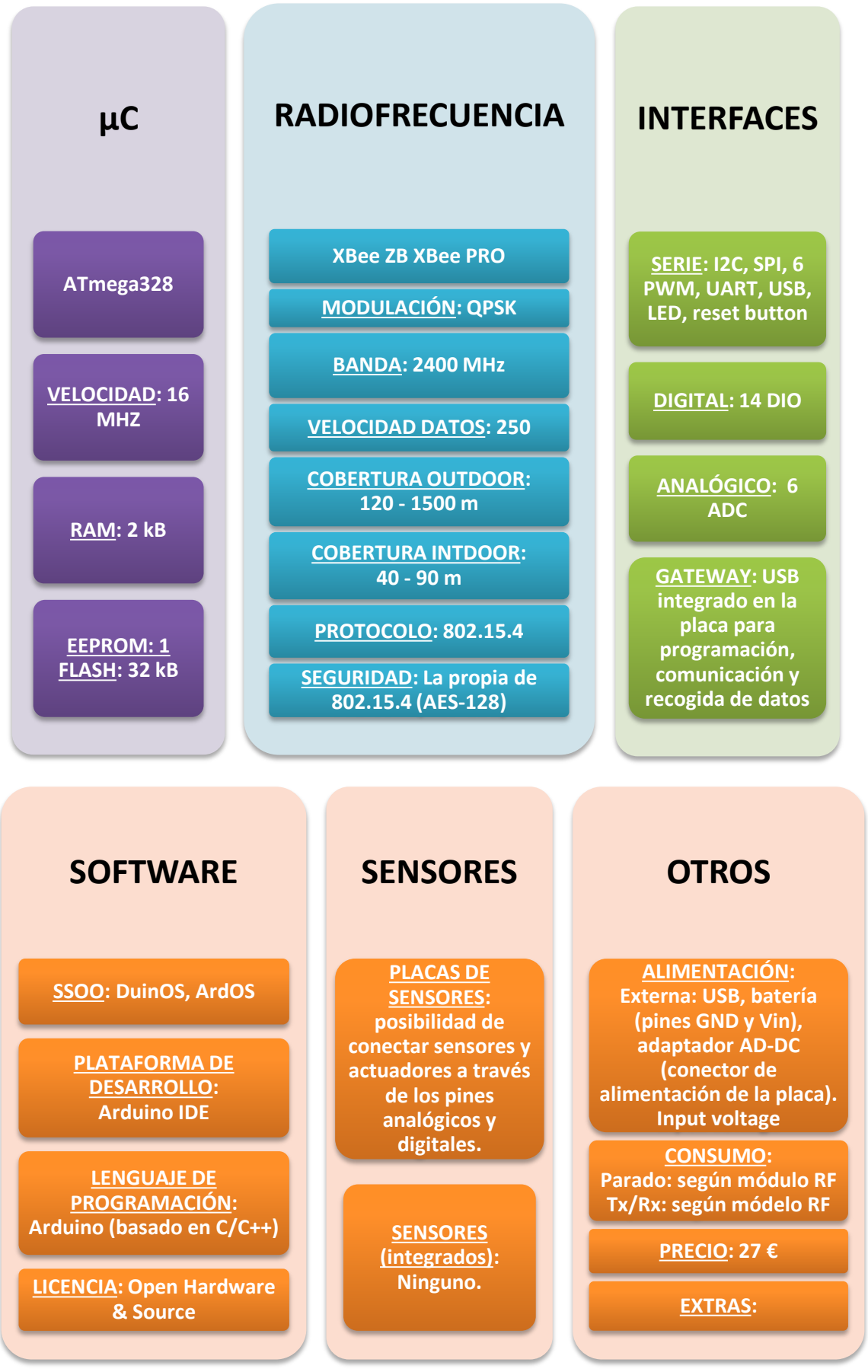


Figura 27. ArduinoUNO


µC
ATmega328
VELOCIDAD: 16 MHZ
RAM: 2 kB
EEPROM: 1
FLASH: 32 kB
RADIOFRECUENCIA
XBee ZB XBee PRO
MODULACIÓN: QPSK
BANDA: 2400 MHz
VELOCIDAD DATOS: 250
COBERTURA OUTDOOR: 120 - 1500 m
COBERTURA INTDOOR: 40 - 90 m
PROTOCOLO: 802.15.4
SEGURIDAD: La propia de 802.15.4 (AES-128)
INTERFACES
SERIE: I2C, SPI, 6 PWM, UART, USB, LED, reset button
DIGITAL: 14 DIO
ANALÓGICO: 6 ADC
GATEWAY: USB integrado en la placa para programación, comunicación y recogida de datos
SOFTWARE
SSOO: DuinOS, ArdOS
PLATAFORMA DE DESARROLLO: Arduino IDE
LENGUAJE DE PROGRAMACIÓN: Arduino (basado en C/C++)
LICENCIA: Open Hardware & Source
SENSORES
PLACAS DE SENSORES: posibilidad de conectar sensores y actuadores a través de los pines analógicos y digitales.
SENSORES (integrados): Ninguno.
OTROS
ALIMENTACIÓN: Externa: USB, batería (pines GND y Vin), adaptador AD-DC (conector de alimentación de la placa). Input voltage
CONSUMO: Parado: según módulo RF Tx/Rx: según módelo RF
PRECIO: 27 €
EXTRAS:

2.8. Estudio de Mercado

2.8.1. Justificación de parámetros escogidos

Es necesario, a la hora de escoger una plataforma hardware, tener presente ciertos aspectos importantes tanto a nivel hardware como software.

Para empezar es necesario fijarse en los sensores integrados en la propia placa y en las posibles placas de sensores que podrán conectarse con ésta. En función de las aplicaciones que se deseen desarrollar se requerirán unos u otros sensores, capaces de recoger la información requerida de la zona de monitorización.

Desde el punto de vista software, es necesario tener en cuenta la plataforma de desarrollo que soportará una determinada placa, puesto que esta herramienta facilitará la tarea de programación a los desarrolladores de aplicaciones.

Por otro lado, debe tenerse en cuenta que los recursos hardware de los nodos de las WSANs son limitados. Esto hace que se deba tomar en consideración algunos parámetros como:

- **Velocidad del microcontrolador.** Al ser más elevada se podrán ejecutar mayor número de instrucciones por segundo, lo que provocará un mejor rendimiento y eficiencia del sistema.
- **Memorias ROM y Flash.** Cuanta más capacidad de almacenamiento tengan, se permitirá guardar mayor cantidad de datos y aplicaciones.
- **Memoria RAM.** Es un dato a tener en cuenta puesto que cuanta más capacidad tenga se podrán ejecutar mayor número de aplicaciones y más pesadas al mismo tiempo.
- **Sistema operativo.** La baja capacidad de las memorias obliga a escoger sistemas operativos de baja capacidad y destinados específicamente a WSANs. Éste además influirá en el lenguaje de programación y en la plataforma de desarrollo que se utilizarán para la creación de aplicaciones.
- **Rango de cobertura.** Es importante puesto que la baja capacidad del módulo de radiofrecuencia limitará la cobertura.
- **Fuente de alimentación.** Es necesario conocer el tipo de baterías y/u otras posibles fuentes de alimentación, puesto que es una de las principales limitaciones de las WSANs.

A su vez, el protocolo de comunicación que soporta la plataforma hardware influye de manera directa sobre el tiempo de respuesta en la toma de decisiones por parte de los actuadores ante la información que le ofrecen los sensores. En ciertas aplicaciones como vigilancia y seguridad es importante una baja latencia de la red que permita asegurar la correcta realización.

No solo el protocolo influye en el tiempo de respuesta, sino también en la seguridad y privacidad de las WSANs. Al tratarse de protocolos inalámbricos la información transmitida es más vulnerable que en otro tipo de redes, con la contrapartida añadida de que los algoritmos de seguridad suponen una carga computacional y de almacenamiento elevada para un sistema de baja capacidad.

Así mismo, se debe tener en cuenta la interconectividad necesaria entre la WSAN y otras redes de datos externas. Para ello se han definido dispositivos que actuarán como pasarela entre ambas redes, llamados gateways. Se utilizarán las interfaces de las placas para conectar al gateway con la estación base.

Otra de las características de las WSANs es el rango de cobertura que vendrá determinado por el protocolo de comunicación inalámbrico utilizado. En función de las aplicaciones que se pretenden desarrollar, el rango de cobertura requerido para la WSAN será diferente. No tendrá las mismas necesidades de cobertura una aplicación que se despliega en un bosque, para control de incendios, que la gestión de un Smart Building.

Por último, una de las características más importantes en las WSANs es el tiempo de vida útil de los nodos, el cual está determinado por los siguientes factores:

- **Fuente de alimentación.** Es necesario conocer de qué manera es suministrada la energía a la placa.
- **Consumo del módulo de radiofrecuencia.** Los esquemas de modulación, la velocidad de transferencia de datos, la banda de frecuencia y el protocolo de comunicaciones utilizados influyen directamente sobre el tiempo de vida útil de la placa, siendo el transceptor el elemento que más consume del nodo.
- **Consumo del microcontrolador.** El modo de operación del procesador y el sistema operativo es una característica crucial para alargar la vida útil.

2.8.2. Tabla Comparativa de Plataformas Hardware Dominantes en el Mercado Actual

Además de las plataformas hardware ya enunciadas en el Apartado 2.7 se tuvieron en cuenta otras motas disponibles en el mercado que quedan recogidas y organizadas en una tabla comparativa atendiendo a los diversos parámetros que han sido considerados a lo largo de este documento y justificados en el Apartado 2.8.1. La tabla comparativa se encuentra en el Anexo I.

2.8.3. Conclusión

El objetivo principal de este proyecto es integrar una nueva plataforma hardware en un sistema ya existente, que hace uso de la mota SunSPOT. Para alcanzar este objetivo se trabajará en las comunicaciones y el middleware ya desarrollado.

Como se ha mostrado en el documento existe una gran variedad de plataformas hardware en el mercado, de entre las cuales se tendrá que seleccionar la que mejor se adecúe a las necesidades de este proyecto.

Al tratarse de un proyecto de investigación, y no de un proyecto comercial o con vistas a ser desplegados en un entorno concreto, no existen limitaciones tan restrictivas a la hora de seleccionar las especificaciones técnicas de la plataforma. Por ello se ha escogido la plataforma ArduinoUNO, que cubre todas las necesidades de este proyecto, presenta muchos retos tecnológicos al disponer de pocos recursos y se trata de la más económica.

Además, Arduino en el año 2011 había distribuido más de 250.000 unidades, por lo que se puede estimar que en 2013 se alcance un millón de dispositivos vendidos. Ejemplo de ello es que cuenta con una amplia comunidad de desarrolladores, que en el momento en que se realizó este proyecto, alcanzaba los 153.000 miembros en el foro oficial de Arduino. Esto proporciona una gran accesibilidad a la plataforma, contando con una inmensa cantidad de información de múltiples fuentes. [**53**]

Por otro lado, la manera de adquirir datos del entorno (temperatura, sonido, luz, etc.) en Arduino será tan sencilla como hacer uso de las entradas analógicas y digitales (SPI e I2C) cubriendo el 99% de los sensores del mercado, desde los más sencillos y baratos hasta los más complejos y caros. [**54**]

Capítulo 3:

Caso de Uso

3. Caso de Uso

3.1. Diseño de la Solución

En este apartado se pretende introducir y describir a un alto nivel el diseño del sistema de este proyecto ya que en el apartado 3.2 se entrará en un mayor detalle. Para ello se comenzará con una explicación global del diseño llevado a cabo para el sistema planteado en este proyecto, continuando con la presentación de los casos de uso del sistema, los componentes, las clases y diagramas de secuencia de la aplicación. Se expondrán los problemas encontrados a lo largo de la realización de este proyecto y por último se explicará el protocolo propio desarrollado para este sistema.

3.1.1. Diseño del Sistema

El sistema que se pretende diseñar y posteriormente desarrollar sigue los principios de las WSAN (Wireless Sensor and Actuator Network). El objetivo es crear una red inalámbrica de sensores y actuadores que ofrezca dos funcionalidades:

- **Descubrimiento de dispositivos finales.** Se pretende que los dispositivos finales sean capaces de publicar, en forma de mensaje JSON, las características que los definen y los servicios que ofrecen cuando se hayan unido a la red. De manera que cuando el usuario haga uso de la aplicación, reciba dicha información en la pantalla principal y sepa que ya están disponibles los dispositivos para su uso.
- **Envió de peticiones y recepción de respuestas.** Se busca que el usuario pueda desde la aplicación solicitar un servicio a uno de los dispositivos finales disponibles, mediante el uso de peticiones y respuestas. Se han completado dos posibles servicios:
  - ✓ **Obtención de la temperatura en un entorno determinado:** un sensor de temperatura situado en uno de los dispositivos finales estará realizando mediciones bajo demanda. Cuando el usuario solicite este servicio, podrá pedir la medida en Celsius o Fahrenheit y le será mostrado el resultado en la pantalla principal.
  - ✓ **Encender o apagar un LED:** en el otro dispositivo final se ha ubicado un LED, que espera recibir peticiones indicando el encendido o apagado de éste. El usuario podrá decidir el estado del LED.

El sistema estará formado por un PC y tres nodos como se muestra en la Figura 28. Los nodos son placas Arduino UNO (rev3) junto con un módulo XBee de radiofrecuencia. Uno de ellos actuará como coordinador de la red WSAN y los dos restantes ejercerán de dispositivos finales. Uno de los dispositivos finales actuará como nodo sensor, tomando medidas de temperatura, y el otro operará como un nodo actuador sobre un LED.

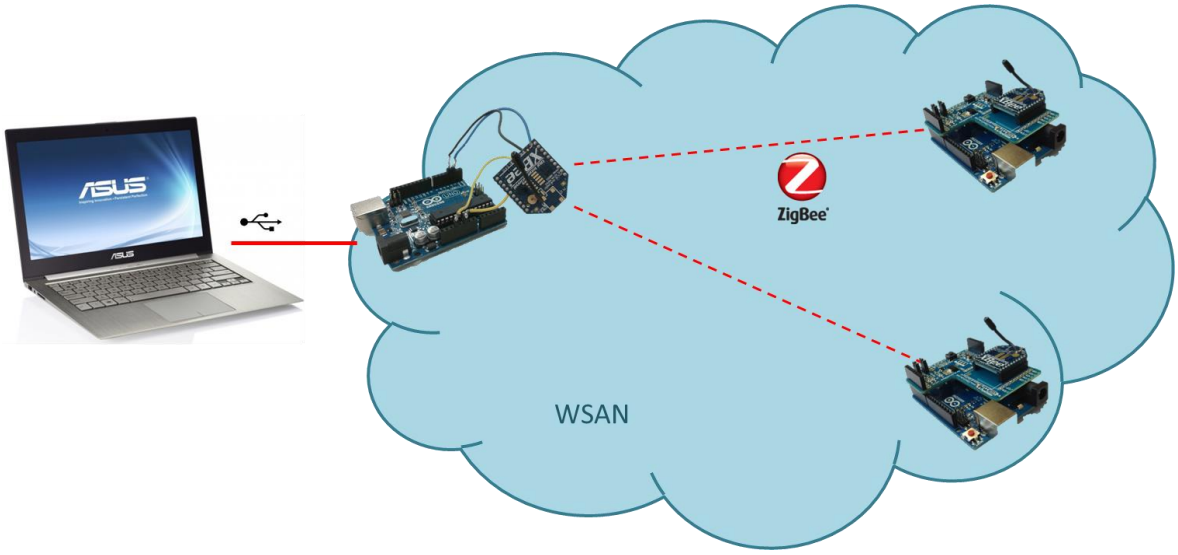


Figura 28. Esquema del sistema completo.

Se decidió implementar la aplicación en el lenguaje de programación Java ya que ofrece soporte a multitud de plataformas y una gran facilidad para la publicación de servicios web.

3.1.2. Caso de Uso del Sistema

En el sistema se identifican tres actores: el usuario, el coordinador y los dispositivos finales.

- **Usuario.** Persona que hace uso de la aplicación.
- **Coordinador.** Mota conectada al PC que actúa como gateway entre el PC y los dispositivos finales.
- **Dispositivos Finales.** Mota sensor o actuador que proporciona información de interés para el usuario o actúa sobre un dispositivo de la manera que el usuario decida.

Usuario. Será capaz desde un PC, a través de la aplicación diseñada en este proyecto, de acceder a los servicios ofrecidos por nodos remotos en una red inalámbrica de sensores

y actuadores. Cuando el usuario ejecute la aplicación, irán apareciendo en la pantalla principal de ésta los dispositivos finales disponibles en la red WSAN. Esto es posible gracias al descubrimiento automático de dispositivos finales. Éstos, al unirse a la red WSAN, publicarán sus características y servicios por medio de un mensaje JSON que enviarán al coordinador y éste a su vez al PC.

Una vez que el usuario disponga en pantalla de la lista de dispositivos disponibles podrá realizar ciertas acciones sobre ellos, como se puede observar en la Figura 29.

- **Asignar "friendly name".** En la pantalla principal los dispositivos se mostrarán en un árbol y cada uno de ellos se podrá desplegar mostrando todas las características y los servicios propios de cada uno de ellos. El nombre que aparece por defecto es el identificador del dispositivo que se corresponde con la dirección de 64 bit única para cada uno de ellos. Para facilitar la identificación de los dispositivos por parte del usuario y que sea más amigable la interfaz se ofrece la posibilidad de modificar dicho nombre por un "friendly name".
- **Realizar una petición.** Como ya se ha comentado, se busca que el usuario de manera sencilla sea capaz de solicitar servicios y obtener la respuesta en un PC. Esta acción se llevará a cabo de manera muy simple y accesible para el usuario, teniendo únicamente que seleccionar uno de los dispositivos y a continuación, pulsar un botón de "Petición". Dependiendo del dispositivo el siguiente paso será distinto, por ejemplo en el caso del dispositivo que opera como sensor aparecerá una ventana dando la opción de obtener la temperatura en Celsius o Fahrenheit con un botón de "Obtener temperatura" y un cuadro donde se mostrara el resultado.
- **Eliminar un dispositivo final.** También se dará la opción al usuario de eliminar cualquiera de los dispositivos finales disponibles de la aplicación. Y como en el resto de casos, se hará de una manera muy intuitiva. Solo tendrá que seleccionar uno de los dispositivos mostrados en el árbol y seguidamente pulsar el botón "Borrar".

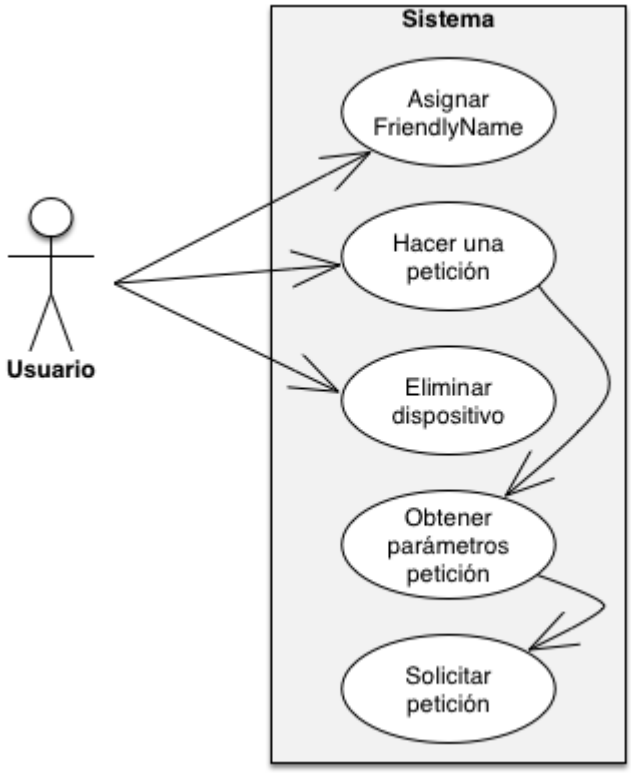


Figura 29. Diagrama de Caso de Uso UML Usuario.

La red inalámbrica de sensores y actuadores de este sistema, como ya se ha comentado, está formada por un coordinador y dos dispositivos finales.

Coordinador. Nodo que actuará como gateway entre el PC y los dispositivos finales, por lo tanto se encargará de gestionar las peticiones enviadas desde el PC hacia los dispositivos finales y las respuestas enviadas desde los dispositivos finales hacia el PC.

Dispositivos finales. Nodos que operarán como mota sensor o actuadora y podrán recibir peticiones del PC encaminadas por el coordinador y enviar una contestación en forma de respuesta al coordinador para que éste la encamine al PC. Deberán ser capaces de:

- **Mandar Hello.** Cada dispositivo final publicará, en forma de mensaje JSON, las características que los definen y los servicios que ofrecen cuando dicho dispositivo se una a la red. Seguidamente se fragmenta el mensaje JSON en paquetes de tipo Hello y se envían al coordinador que a su vez lo encaminará al PC. Una vez se reensamble el mensaje JSON en el PC, el usuario visualizará estas características y servicios de dicho dispositivo en forma de árbol en la pantalla.

- **Mandar Response.** Para enviar un paquete de tipo Response se siguen los siguiente pasos:
  - ✓ Procesar el paquete de petición recibido.
  - ✓ Obtener la información solicitada por el usuario de sus sensores o realizar determinada acción seleccionada por el usuario sobre un actuador.
  - ✓ En el caso del nodo sensor, crear un paquete de respuesta con la información solicitada y enviarlo al coordinador.

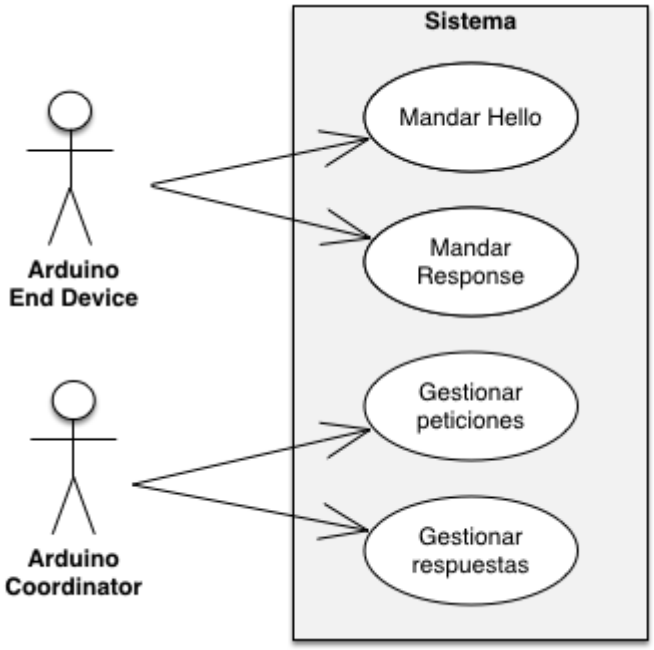


Figura 30. Diagrama de Caso de Uso UML Nodos (Coordinador y Dispositivo Final).

3.1.3. Componentes de la Aplicación

En la Figura 31 se muestra el diagrama de componentes de la aplicación, que se ejecuta en el PC y con la que interactúa el usuario, desarrollada en este proyecto. El diseño de la aplicación se hizo teniendo en cuenta estos tres componentes:

- **IUsuario.** Es la parte central de la aplicación, en él se encuentra la clase principal, la interfaz de usuario y la gestión de eventos.
- **Comunicaciones.** Es el componente encargado de las comunicaciones de la aplicación entre el PC y la red WSAN.

- **Gestor de dispositivos.** Componente que gestiona el conjunto de dispositivos finales de la red inalámbrica de sensores y actuadores.

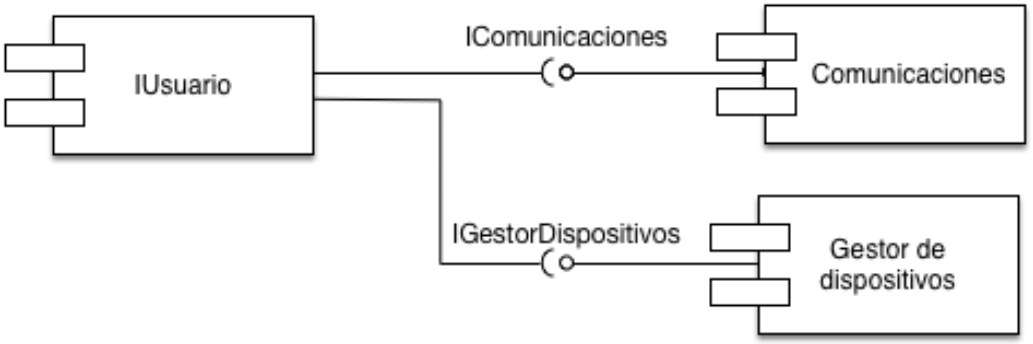


Figura 31. Diagrama de Componentes UML

3.1.4. Clases de Objeto de la Aplicación

A continuación se van mostrar los diagramas de clases UML de cada uno de los componentes que conforman la aplicación que se ejecuta en el PC y con la que interactúa el usuario. Se dará una breve explicación ya que en el apartado 3.2.4.4 se describirán con detalle cada una de las clases y todos sus métodos.

Clases del componente IUsuario. La representación de las clases se ha dividido en dos diagramas. Ambos parten de la clase principal App de la aplicación. En la Figura 32 se puede ver el primer diagrama que contempla la dependencia con la interfaz gráfica siendo estas clases *WMain*, *WTemperatureSensor*, *WLed*, *WProgressBar* y la asociación con la clase intermedia *EventController* que permite la separación de los datos y la lógica de la interfaz gráfica.

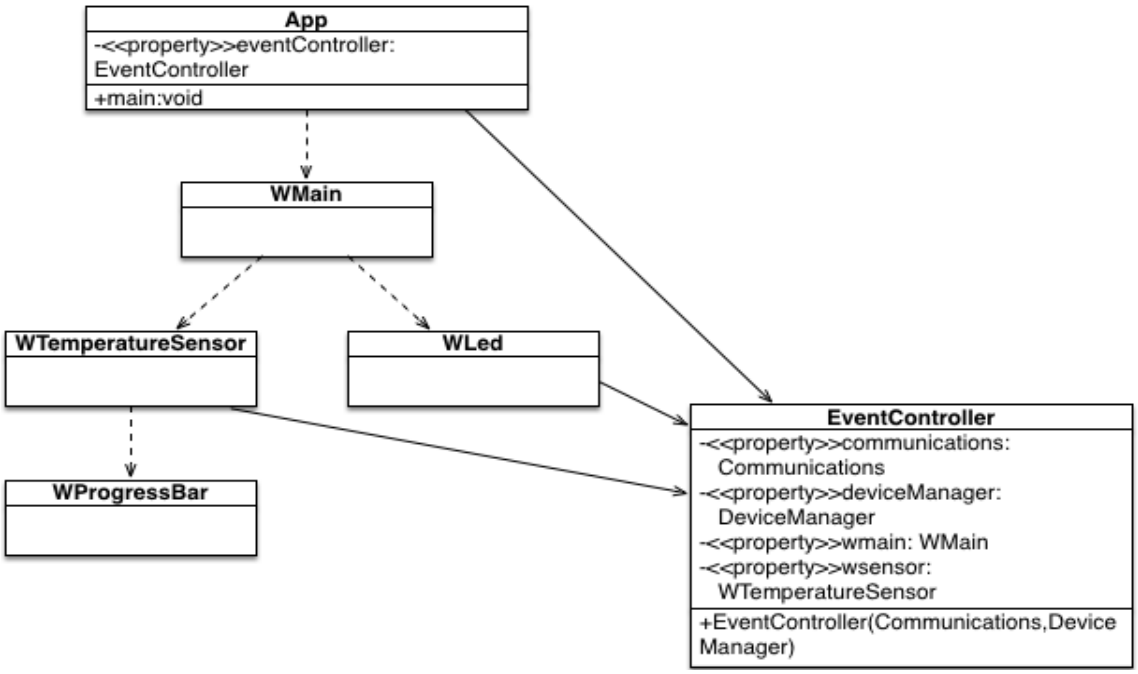


Figura 32. Diagrama de Clases UML del componente IUsuario GUI

En el segundo diagrama de la Figura 33 se muestra la relación con la clase abstracta *EventManager* la cual se encarga de gestionar los eventos que se producen en la aplicación. De ésta heredan los dos suscritores existentes hasta el momento en la aplicación, *HelloProcess* y *ResponseProcess*, una encargada de los eventos relacionados con el descubrimiento de dispositivos finales y la otra con el procesado de las respuestas provenientes de los dispositivos finales. Éstas últimas también tienen una relación de asociación con la clase *EventController*.

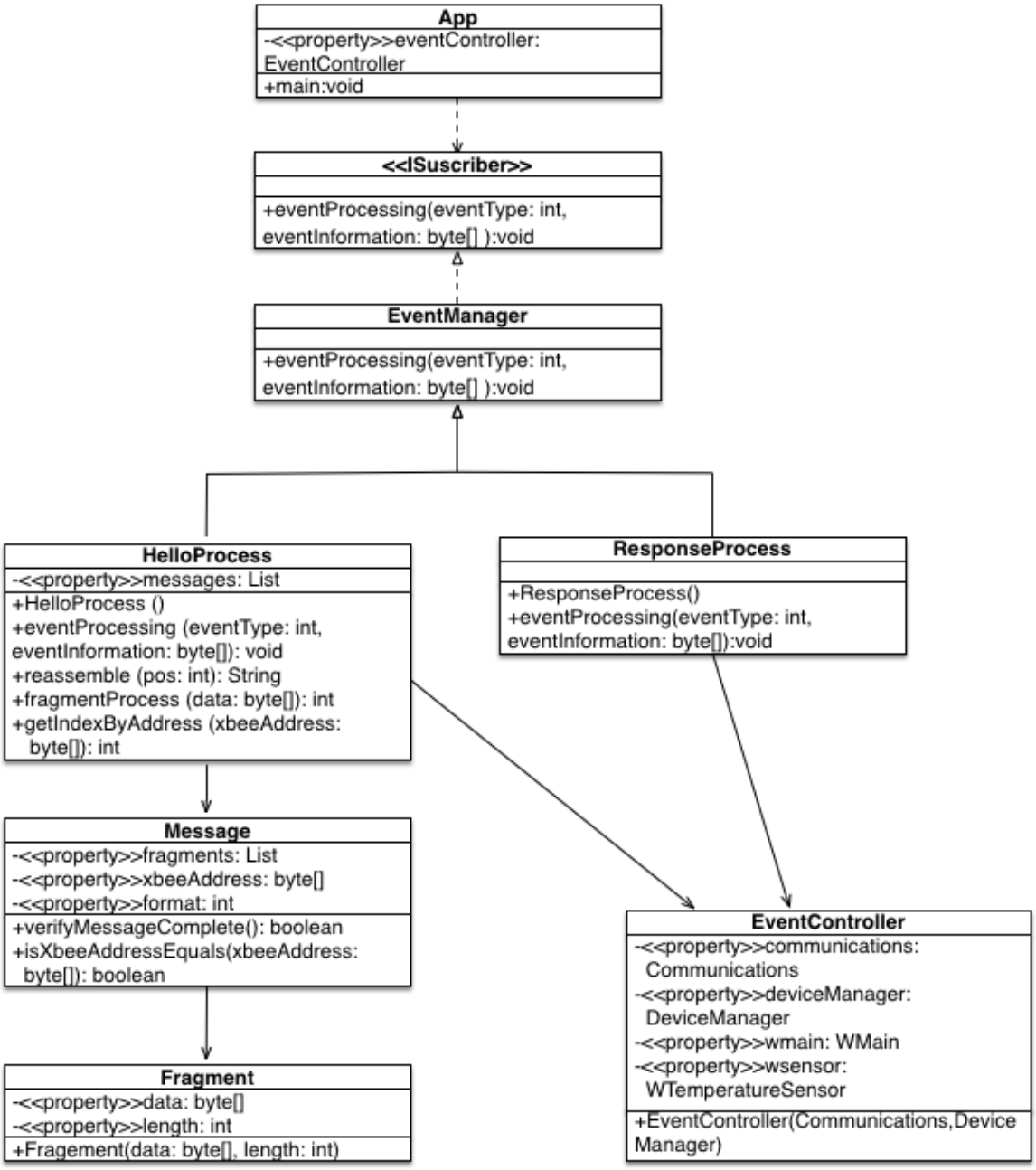


Figura 33.Diagrama de Clases UML del componente IUsuario Lógica

Clases del componente Comunicaciones. La clase principal de este componente es *Communications*, la cual es la encargada de las comunicaciones entre el PC y el coordinador. Para ello tendrá que establecerse una comunicación serie entre éstos, siendo la clase *SerialCommunications* la que tenga ese cometido. También tendrá que tener relación con la clase *EventManager*, puesto que según llegue un paquete y se averigüe el tipo de evento del que se trata, habrá que enviar dicho paquete al suscriptor

correspondiente. La relación entre suscriptor y evento se almacenará en la clase *SuscriberEventRealtionship*.

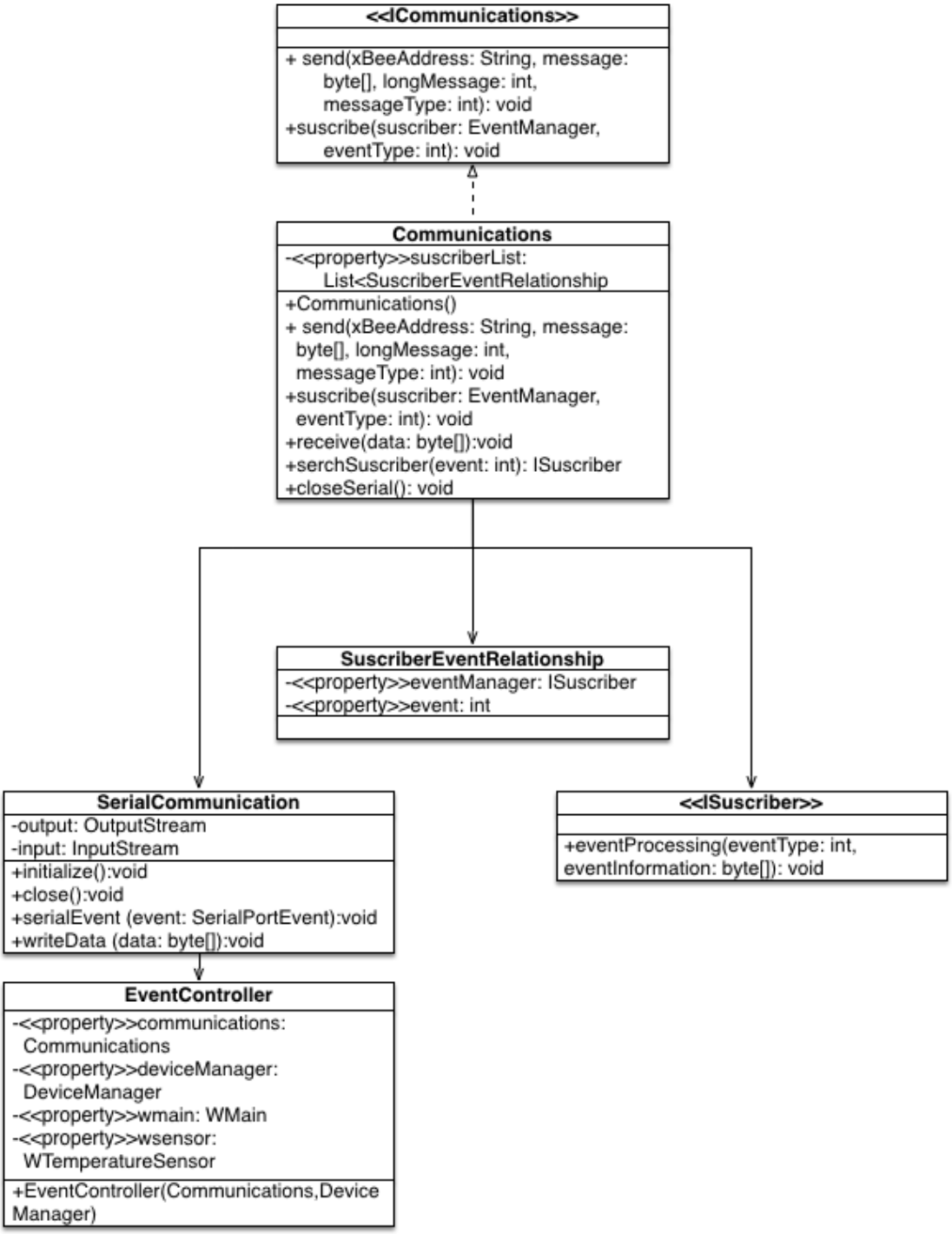


Figura 34.Diagrama de Clases UML del componente Comunicaciones

Clases del componente Gestor de dispositivos. *DeviceManager* es la clase principal y se encarga de gestionar los dispositivos finales de la red inalámbrica de sensores. Como ya se ha comentado en el descubrimiento de dispositivos finales, éstos publican un mensaje JSON, siendo la encargada de procesarlo la clase *JSON*. Una vez procesado se podrá almacenar el nuevo dispositivo en el conjunto de dispositivos finales. También existe una relación con la clase *EventController* pues será necesario actualizar la interfaz gráfica cada vez que se añada o elimine un dispositivo.

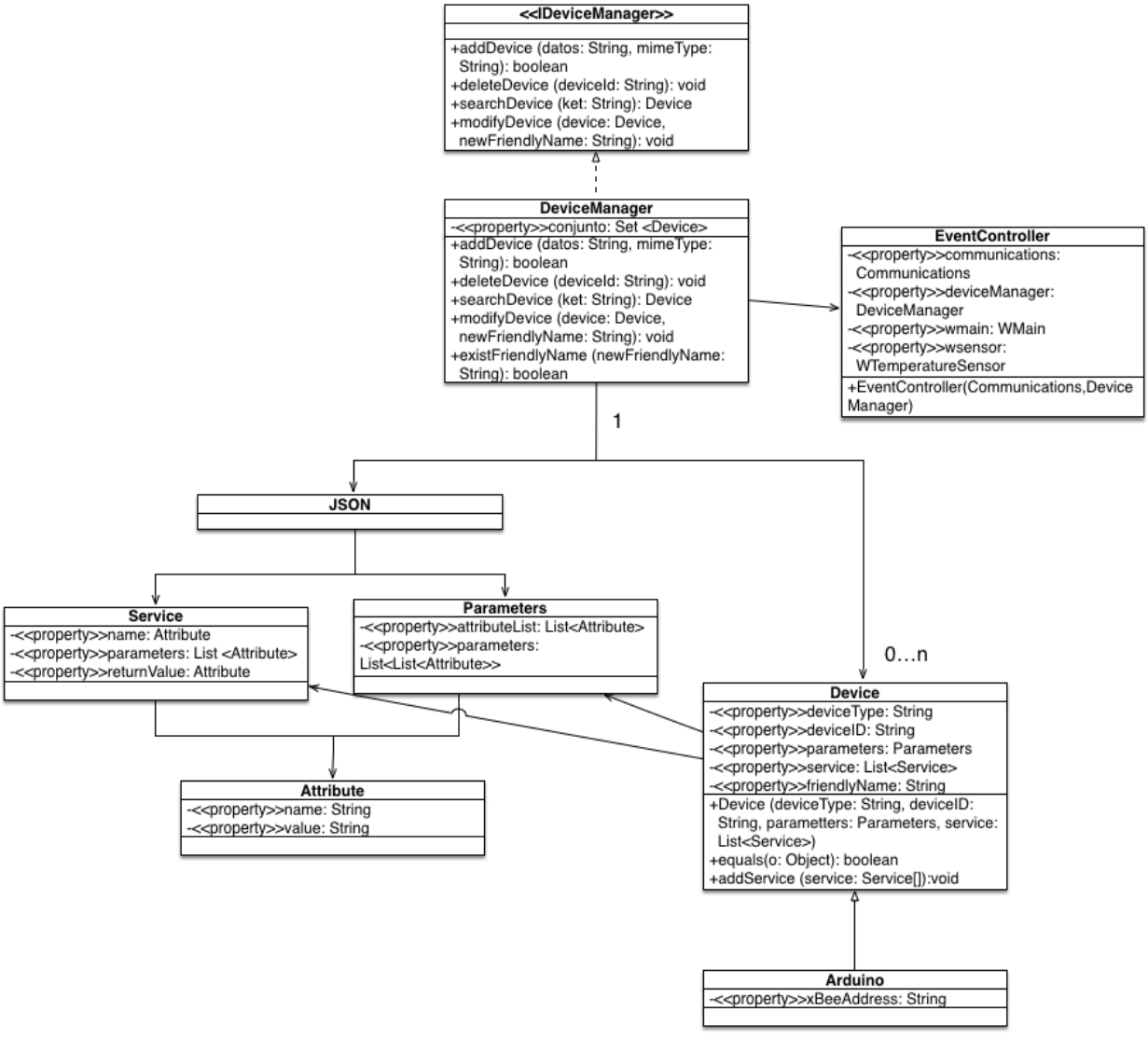


Figura 35. Diagrama de Clases UML del componente Gestor de Dispositivos

3.1.5. Secuencia de la Aplicación

Se ha realizado el diagrama de secuencia de la Figura 36 para tener una visión general del funcionamiento y de las interacciones entre los objetos de la aplicación. En la clase principal por un lado se creará la interfaz gráfica, mostrando la pantalla principal de la aplicación, y los objetos de las clases *Comunications*, *DeviceManager*, *HelloProcess*, *ResponseProcess*. Dichas instancias, junto con las de *WMain* y *WTemperatureSensor*, se almacenarán en la clase *EventController*, permitiendo el acceso a los métodos desde cualquier clase ya que se definió estática. Seguidamente *HelloProcess* y *ResponseProcess* se suscribirán indicando el tipo de evento del que se encargan.

Llegados a ese punto, la aplicación está preparada para la recepción de paquetes "Hello" para el descubrimiento de dispositivos finales. Estos irán llegando según los dispositivos se vayan uniendo a la red inalámbrica de sensores y será la clase *Communications* la que envié los paquetes al suscritor *HelloProcess* encargado del procesado de éstos. La clase *DeviceManager* junto con la clase *JSON* parsearán el mensaje JSON recibido y se añadirán el dispositivo final al conjunto.

A partir de aquí el usuario puede solicitar un servicio de cualquiera de los dispositivos finales. La aplicación enviará una petición al coordinador quien a su vez la encaminará al dispositivo final correspondiente. El dispositivo final deberá enviar una respuesta hacia el coordinador y este encaminarla hacia el PC. La aplicación procesará el paquete de respuesta mostrando por pantalla los resultados solicitados por el usuario. En el caso de que el dispositivo final opere como nodo actuador simplemente se realizarán las acciones solicitadas por el usuario y no se recibirá paquete de respuesta.

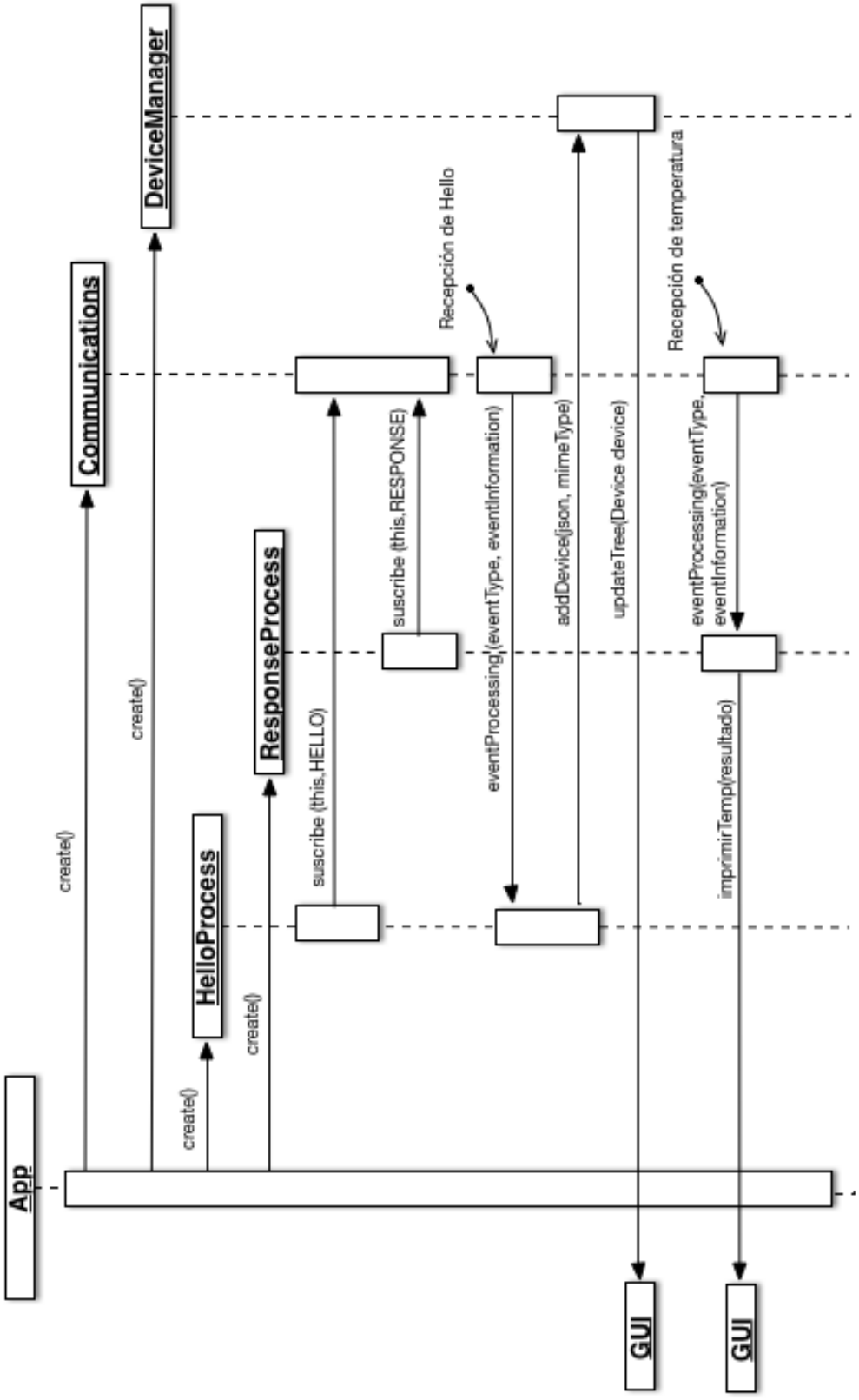


Figura 36. Diagrama de secuencia UML de la aplicación.

3.1.6. Problemas Encontrados Durante la Realización del Sistema

A lo largo del desarrollo del sistema se han planteado ciertos problemas que serán expuestos en los siguientes apartados. También se explicará la decisión tomada para dar solución a estas dificultades.

3.1.6.1. Comunicación Serie PC-Arduino y Viceversa.

El sistema planteado en el proyecto está formado por un PC y por una red inalámbrica de sensores y actuadores. Éstos han de poder comunicarse entre sí y para ello será necesario establecer una comunicación serie PC- placa Arduino UNO (rev3).

Teniendo en cuenta que la aplicación está implementada en el lenguaje de programación Java, se buscaron posibles librerías que permitieran esta comunicación serie.

Debido a la independencia de Java respecto a las plataformas hardware, ofrecer soporte para la interfaz serie es complicado, ya que sería necesario un API estandarizado con implementaciones específicas para cada una de ellas. Sun Microsystems desarrolló un API para comunicaciones serie llamado JavaComm, pero debido a que solo soportaba Solaris SPARC, Solaris x86 y Linux x86 y que no da soporte actualmente, se ha decidido buscar una solución de terceros. Tras valorar las posibilidades existentes se eligió trabajar con la librería RXTX, que será explicada en el apartado 3.2.3.3.

3.1.6.2. Necesidad de Varios Puertos Serie en el Coordinador

Una vez resuelto el problema de la comunicación serie se plantea uno nuevo a consecuencia de éste último, la necesidad de varios puertos serie en la placa Arduino UNO (rev3) que ejerce de coordinador. Uno para la comunicación serie entre PC-coordinador y otro para la comunicación coordinador-dispositivos finales vía módulo XBee de radiofrecuencia.

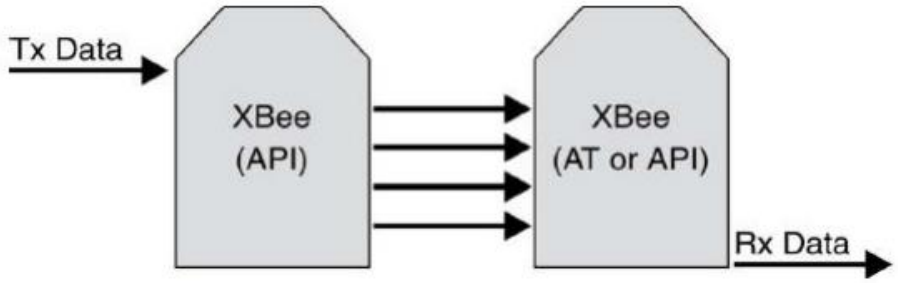
El Arduino UNO es una placa que facilita numerosas maneras de comunicación con un PC, otro Arduino u otros microcontroladores. Está basada en el microcontrolador Atmega328 que ofrece soporte para las comunicaciones serie a través de los pines 0 (RX) y 1 (TX) y al contar con el Atmega 16U2 programado como convertidor USB-to-serial, canalizará la comunicación serie sobre el USB y aparecerá como un puerto COM virtual en el software del PC [**55**]. Esto da solución a la primera de las comunicaciones que son necesarias, PC-Arduino.

Ahora hay que encontrar una solución para dar apoyo a la comunicación entre coordinador y dispositivos finales a través del módulo XBee. Para ello se decidió utilizar la librería para Arduinos llamada ***SoftwareSerial*** (o ***NewSoftSerial***), la cual ha sido desarrollada para permitir la comunicación serie a través de cualquiera de los pines digitales de la placa Arduino, utilizando el software para replicar la funcionalidad. El montaje y utilización de ésta en el proyecto se explicará de manera más detallada en los apartados 3.2.2.1 y 3.2.3.4 respectivamente.

3.1.6.3. Fragmentación

Como se comentó en el apartado 3.1.1 de diseño del sistema, en este proyecto se pretende que cada uno de los dispositivos finales que formen la red inalámbrica de sensores y actuadores publique las características que los definen y los servicios que proporcionan. Para esto se utiliza un mensaje JSON de alrededor de unos 256 caracteres (bytes).

Una transmisión ZigBee puede soportar hasta 84 bytes de carga útil (se verá reducida si se utiliza encriptación). Sin embargo, el firmware XBee ZB de los módulos XBee de radiofrecuencia utilizado en este proyecto para la comunicación inalámbrica soporta una función denominada fragmentación que permite que un solo paquete de datos de gran tamaño se pueda dividir en múltiples transmisiones y reensamblarlas en el receptor antes de entregarlo al Arduino.



La trama de transmisión API puede incluir hasta 255 bytes de datos, los cuales se dividirán en múltiples transmisiones y se reensamblarán en el lado receptor. Si uno o más de los mensajes fragmentados no se reciben en el receptor, éste descartará el mensaje completo, y el remitente indicará fallo en la transmisión en la trama API de tipo ZigBee Transmit Status.

A pesar de que esta funcionalidad de fragmentación ofrecida por el firmware XBee ZB podría ser suficiente para éste proyecto, se ha decidido no utilizarla y hacer una

implementación para la fragmentación, asegurándonos de que el envío no se vea restringido por el tamaño de 255 bytes que permite el modo API del XBee. Se pretende que el coordinador sea el encargado de esta tarea de reensamblaje y fragmentación, pero debido a la escasez de memoria en la placa Arduino UNO (rev3) que se explica en el apartado 3.1.6.4 fue y es imposible llevar a cabo este planteamiento.

Finalmente se tomó la decisión de trasladar la función de reensamblado a un dispositivo con mayor capacidad, el ordenador. Para llevarlo a cabo se diseñó un protocolo E2E (End-to-End), explicado en el apartado 3.1.7, que será utilizado entre dispositivos finales y el ordenador. La implementación hecha en el PC para esta funcionalidad se describe en el apartado 3.2.4.4.

3.1.6.4. Escasez de Memoria

Existen tres tipos de memoria en el microcontrolador (ATmega328) utilizado por la placa Arduino UNO (rev3):

- **Memoria Flash (espacio del programa).** Donde se almacena el sketch.
- **SRAM (Static Random Memory, memoria estática de acceso aleatorio)**. Donde los sketches crean y manipulan variables cuando están en ejecución.
- **EEPROM.** Es espacio de memoria que los programadores pueden usar para almacenar información a largo plazo.

La memoria Flash y EEPROM son no-volátiles (la información se mantiene en ellas después de ser apagado). La SRAM es volátil y se perderá la información al apagarlo.

El chip ATmega328 tienen las siguientes características:

Memoria Flash	32 KB (0.5 KB usados por bootloader)
SRAM	2 KB
EEPROM	1 KB

Tabla 1. Memorias Arduino UNO (rev3)

Las variables normalmente se almacenan en la SRAM y como se puede apreciar en la Tabla 1, este tipo de memoria en la placa Arduino UNO (rev3) es sólo de 2KB.

En un primer momento se pensó que el coordinador fuese el encargado de la tarea de fragmentación y reensamblado. Se realizaron una serie de cálculos para comprobar las dimensiones de estas estructuras y se observó que era insuficiente el espacio disponible

para almacenar dichas estructuras con sus respectivos datos. Se analizaron posibles soluciones como es la utilización del modificador de variable PROGMEM en los programas de Arduino, el cual permite almacenar información en la memoria flash en vez de en la SRAM. Pero la información almacenada debía ser estática y esto seguía sin ser una solución porque el espacio era aún insuficiente para almacenar toda la lógica.

Finalmente, se llegó a la conclusión de que la capacidad de las placas Arduino UNO (rev3) es muy reducida para lo que se pretendía y se decidió trasladar el proceso de fragmentación a un dispositivo con mayor capacidad, el PC, tal y como ya se comentó en el apartado 3.1.6.3.

3.1.7. Protocolo del Sistema

Como se ha visto en el apartado 3.1.6.3 será necesario realizar fragmentación, por este motivo se define un protocolo End-to-End entre dispositivos finales y el PC como se muestra la Figura 37.

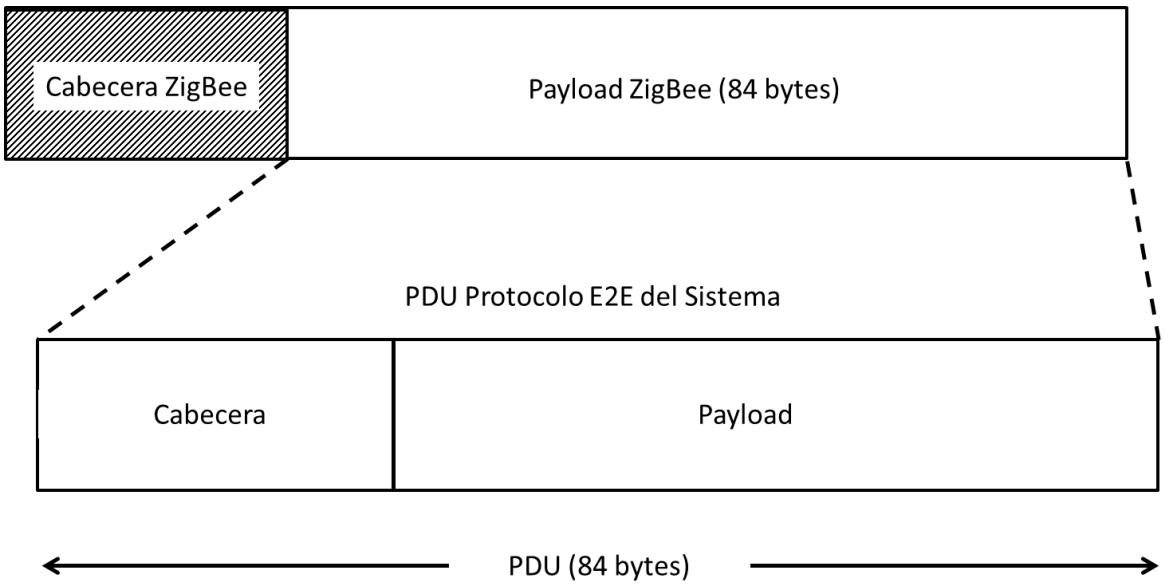


Figura 37. Protocolo E2E del Sistema.

De manera más específica se muestran en la Figura 38 las posibles PDUs del sistema. Se contemplan los tres casos posibles dentro del sistema hasta el momento: HELLO, REQUEST y RESPONSE.

La PDU HELLO facilita el envío de los mensajes JSON por parte de cada uno de los dispositivos finales, en los cuales publican las características que los definen y los servicios proporcionados. Los campos que la forman son:

- Tipo Payload (1 byte): indica el tipo de carga útil que lleva la PDU, en este caso, se corresponde con un 0 (HELLO).
- Número Fragmento (2 bytes): indica el número del fragmento.
- Bit Final (1 byte): indica si se trata del último fragmento del mensaje, en cuyo caso contendrá el valor 1. En caso contrario el valor 0.
- Formato (1 byte): indica el formato del mensaje contenido en la carga útil. En este proyecto solo se contempla el formato JSON, que se corresponde con un 0.

En el caso de la PDU REQUEST & RESPONSE se trata del envío de una petición y una respuesta respectivamente para poder obtener información de un determinado sensor u operar sobre un actuador de un dispositivo final. Para este tipo de PDU solo será necesario el campo "Tipo Payload" que se corresponde con un 1 para peticiones y un 2 para respuestas.

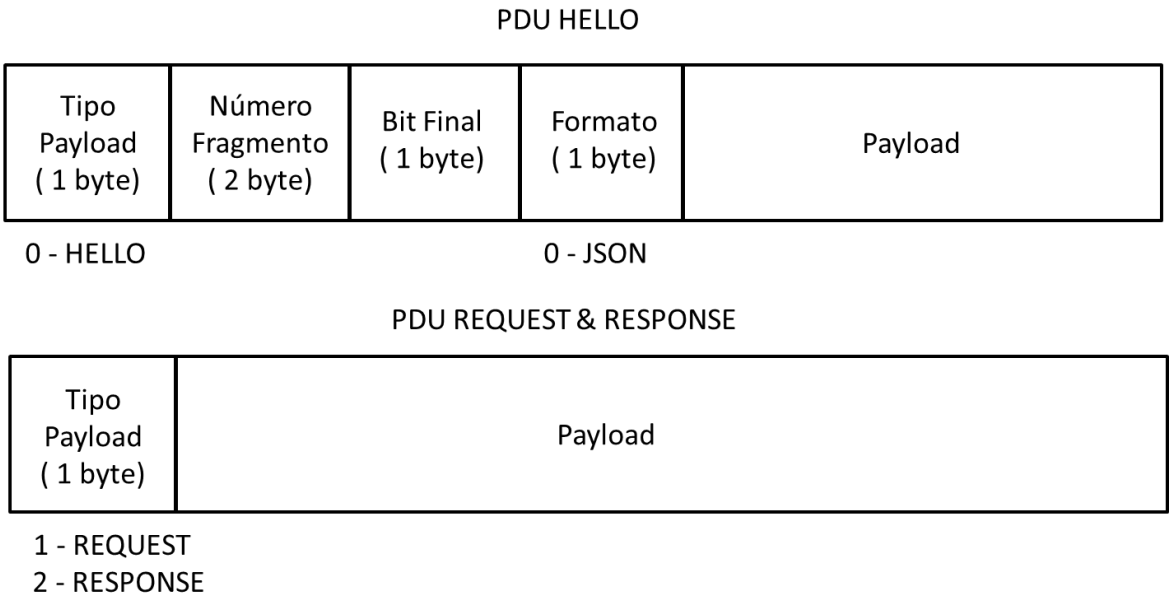


Figura 38. Protocolo E2E del sistema: HELLO, REQUEST y RESPONSE.

3.2. Despliegue

A lo largo de este apartado se describirán las fases llevadas a cabo para poner en funcionamiento el sistema diseñado en este proyecto. Se comenzará con una descripción de los elementos hardware y software utilizados, continuando con los montajes hardware y configuraciones software que se han de llevar a cabo y por último el desarrollo de las distintas implementaciones. Finalmente se facilitará un manual de usuario de la aplicación.

3.2.1. Descripción de Hardware y Software requeridos

3.2.1.1. XBee Shield

La Xbee Shield fue creada por Arduino en colaboración con Libelium, que la desarrolló para su uso en sus motas SquidBee. Permite a la placa Arduino comunicarse de forma inalámbrica utilizando Zigbee. Se basa en el módulo Xbee de la marca MaxStream de la empresa Digi International. [**56**]

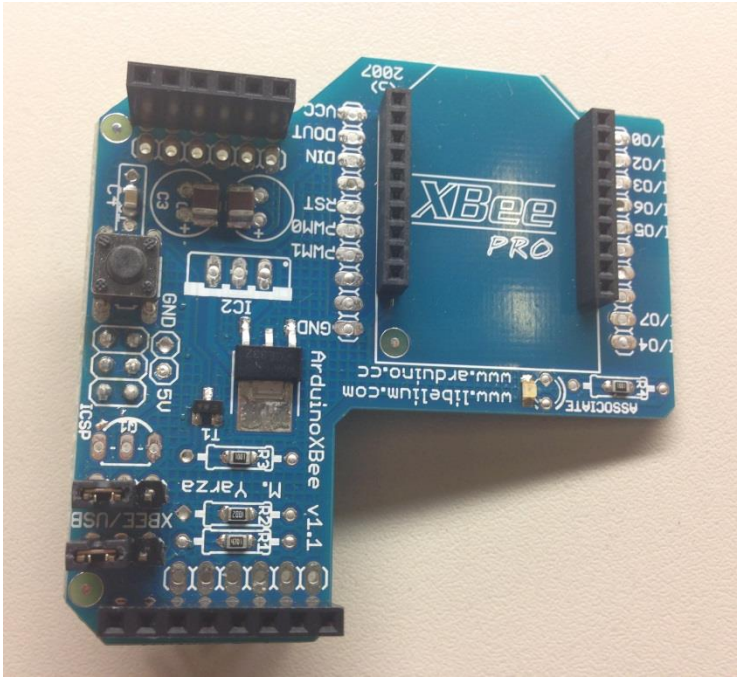


Figura 39. XBee Shield

La Xbee Shield une directamente el módulo Xbee de radiofrecuencia a la placa Arduino, así que ambos comparten el puerto serie. Esto supone un problema a la hora de poder comunicarse con el módulo Xbee de radiofrecuencia y su consiguiente configuración.

La XBee Shield posee dos jumpers que determinan cómo se conecta la comunicación serie del XBee entre el microcontrolador y el chip serie FTDI del Arduino, pudiendo estar en modo USB o modo XBEE [**56**]:

- **Modo XBEE**: en esta posición el pin DOUT del módulo Xbee está conectado al pin RX del microcontrolador y el pin DIN al pin TX. Los pines RX y TX del microcontrolador están todavía conectados a los pines TX y RX del chip FTDI

respectivamente. Es decir, los datos enviados desde el microcontrolador serán transmitidos al ordenador vía USB y a la vez enviados de manera inalámbrica por el módulo Xbee. El microcontrolador SOLO será capaz de recibir los datos desde el módulo Xbee, NO desde el USB del ordenador. Por lo que éste no será programable a través del USB en este modo.

- **Modo USB**: el pin DOUT del módulo Xbee está conectado al pin RX del chip FTDI y el pin DIN al pin TX. Esto significa que el módulo Xbee puede comunicarse directamente con el ordenador. SOLO funciona si el microcontrolador ha sido anulado o quitado de la placa Arduino como se ve a continuación. Si éste se deja en la placa Arduino, solo será capaz de comunicarse con el ordenador vía USB, pero ni el ordenador ni el microcontrolador podrán comunicarse con el módulo Xbee de radiofrecuencia.

Para poder configurar el módulo XBee se tendrá que poner los jumpers en modo USB, permitiendo la conexión directa del XBee con el ordenador; pero como ya se ha comentado, esto SOLO funciona si el microcontrolador ha sido anulado o retirado de la placa Arduino [**57**] [**58**]. Existen diversas maneras de llevar a cabo esto y que solucionarían el problema:

- Sketch vacío: se cargará en el microcontrolador de la placa ArduinoUNO un programa vacío, lo que impide que el gestor de arranque (bootloader) de la placa Arduino responda accidentalmente [**57**].

```
void setup(){}
void loop(){}
```

- Conectar el pin RESET al pin GND con un cable, de esta manera el microcontrolador no se inicia y se puede establecer la comunicación con el Xbee [**59**].
- Retirar el chip ATmega de la placa Arduino utilizando un extractor IC (Integrated Circuit) o un pequeño destornillador de punta plana [**56**].
- Una posible alternativa es utilizar adaptadores USB para Xbee, como pueden ser: SparkFun Xbee Explorer, New Micros Xbee Dongle, Adafruit Xbee Adapter. Éstos permiten la conexión directa con el Xbee [**60**].

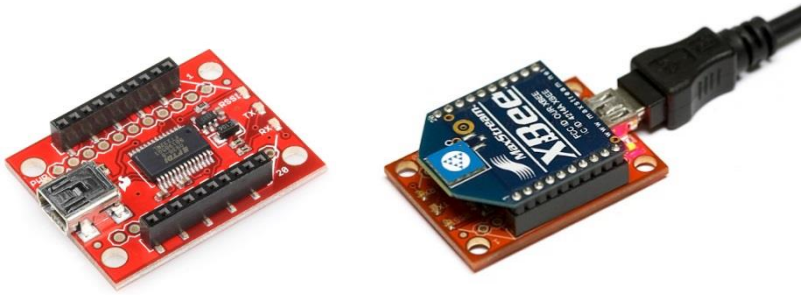


Figura 40. SparkFun XBee Explorer USB [61]

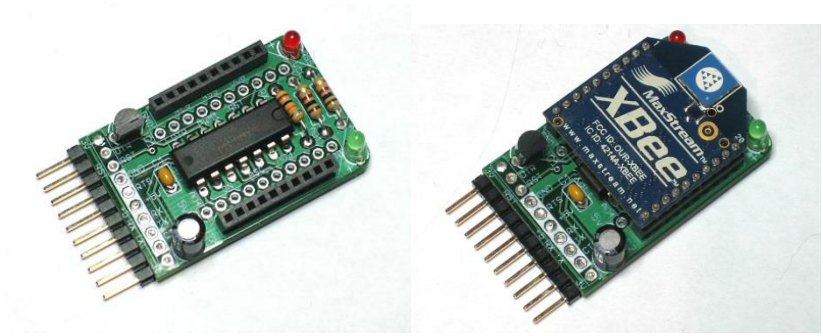


Figura 41. Adafruit XBee Adapter [62]

A lo largo de este proyecto se utilizará la Xbee Shield y se tomará como solución la segunda opción, conectar un cable del pin RESET al pin GND.

3.2.1.2. Módulo XBee de Radiofrecuencia

Figura 42. Módulo XBee Serie 2 con antena cable 2 mW

Xbee es el nombre comercial de Digi International para una familia de módulos de radio que soportan una variedad de protocolos de comunicación. Son módulos de radiofrecuencia fabricados por Digi International que poseen la capacidad de comunicarse

con el microcontrolador a través de la comunicación UART y también tienen pines adicionales que pueden servir para aplicaciones adicionales. Los módulos Xbee proporcionan dos formas de comunicación: transmisión serie transparente (modo AT) y el modo API. Éstos deben ser configurados desde el PC utilizando el programa X-CTU o cualquier software terminal (PuTTY, Hyperterminal, etc.). Para poder usarlos con Arduino se necesita un medio adaptador y en el caso de este proyecto se hará uso de la XBee Shield aunque existen otras posibilidades, como se ha explicado anteriormente.

Existen dos series de estos módulos: la Serie 1 y la Serie 2, también conocida como 2.5. Los módulos de la Serie 1 y la Serie 2 no son compatibles entre sí ya que utilizan distintos chipset y trabajan con protocolos diferentes [**63**].

A lo largo de este proyecto se trabajará con los módulos de la Serie 2, basados en el chipset Ember. Estos módulos mejoran la salida de potencia y los protocolos de datos, permitiendo crear complejas redes malladas basadas en ZigBee. Estos módulos son baratos y de bajo consumo, dos de las principales necesidades de las redes inalámbricas de sensores. Los Xbee Serie 2 proporcionan comunicaciones sencillas y de alta fiabilidad entre los elementos de la red, soportando topologías punto a punto y punto a multi-punto. Las principales características de este módulo son [**64**]:

- ✓ Interior/Exterior: hasta 40 m
- ✓ Al aire libre la línea de visión: hasta los 120 m
- ✓ Potencia de transmisión: 2 mW (+3 dBm)
- ✓ Sensibilidad del receptor: -95 dBm
- ✓ Velocidad de datos RF: 250,000 bps
- ✓ Corriente de transmisiónt: 40 mA (@3.3 V) • Corriente de Recepción: 40 mA (@3.3 V)
- ✓ Corriente con xbee apagado: < 1 μA @ 25ºC

3.2.1.3. X-CTU

Figura 43. Logo X-CTU

X-CTU es el programa de configuración oficial para las radios XBee, proporcionado por Digi International, Inc. Éste solo está disponible para el sistema operativo Microsoft Windows. Se trata de una aplicación diseñada para interactuar con los archivos de firmware que se encuentran en productos de radiofrecuencia de Digi, y para proporcionar una interfaz gráfica de usuario fácil de usar [**65**].

Es posible descargarlo y obtener las instrucciones de instalación de éste en la página oficial de Digi International, Inc [**66**]. Además de X-CTU, si se está utilizando una placa adaptadora (como, Sparkfun XBee USB Breakout board) para los módulos Xbee, también se tendrán que instalar los controladores adecuados para ésta.

- **PC Settings**: se divide en tres áreas básicas, la configuración del puerto COM, la configuración del host y los puertos COM de los usuarios. Permite seleccionar el puerto COM a través del que irá conectada la radio, así como una serie de parámetros de la comunicación serie (Baud Rate, Flow Control, Data Bits, Parity y Stop Bits). Ofrece también la posibilidad de habilitar el modo API y el response timeout [**65**].
- **Range Test**: esta ventana está diseñada para verificar el alcance del enlace radio mediante el envío de un paquete de datos especificado por el usuario y verificando que el paquete de respuesta es el mismo que el que se envió [**65**].
- **Terminal**: con esta ventana se puede interactuar con el modem mediante los comandos AT, para temas relacionados con la configuración [**65**].
- **Modem Configuration**: la pestaña de configuración del módem tiene cuatro funciones básicas [**65**]:
  - ✓ Proporcionar una interfaz gráfica de usuario con el firmware de la radio.
  - ✓ Leer y escribir firmware en el microcontrolador de la radio.
  - ✓ Descargar los archivos de firmware actualizados, ya sea en la web o de un archivo comprimido.
  - ✓ Guardar o cargar un perfil del módem.

3.2.2. Montaje del Hardware

3.2.2.1. Configuración Módulos XBee de Radiofrecuencia

Para poder llevar a cabo la configuración de los módulos XBee, en primer lugar, se deberá conectar en la placa Arduino el pin RESET al pin GND con un cable, que como vimos en el apartado 3.2.1.2 da solución al problema de la conexión directa con el módulo XBee y así su posible configuración. Éste paso únicamente será necesario en este caso.

Figura 44. Conexión pin RESET con GND en ArduinoUNO.

A continuación, ya se puede montar la XBee Shield en el zócalo (socket) de la placa Arduino UNO y el módulo XBee en el zócalo (socket) de la XBee Shield. Como ya se comentó en el apartado 3.2.1.3 la Xbee Shield tiene dos jumpers que determinan como se conecta el módulo Xbee a la placa Arduino UNO, existiendo dos posiciones posibles, la posición XBEE y la posición USB. En este caso, se deben poner los jumpers en la posición USB como se muestra en la Figura 45.

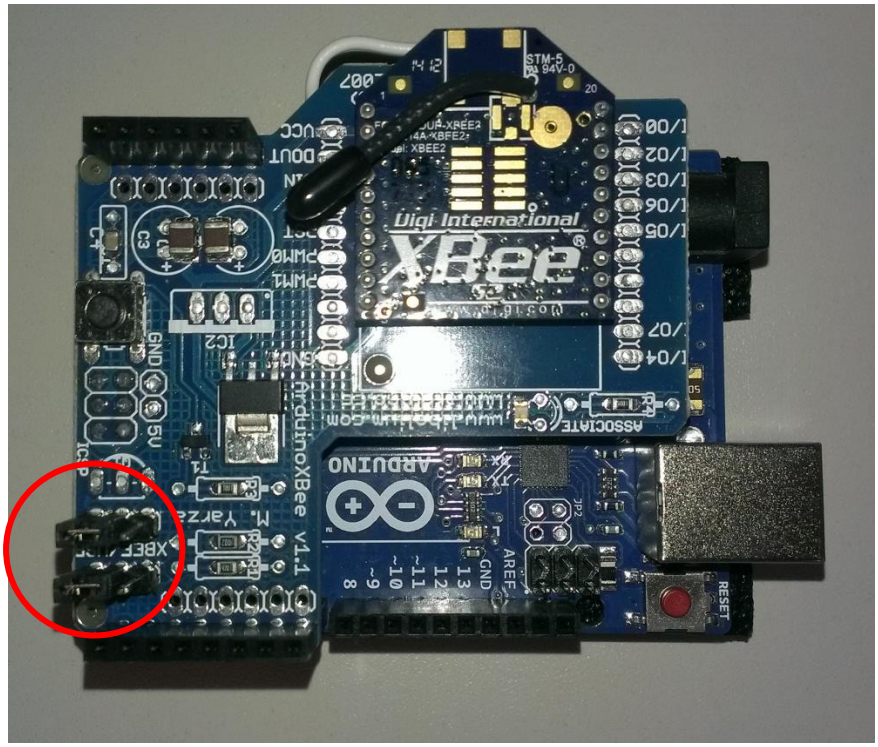


Figura 45. XBee Shield con jumpers en posición USB.

Una vez tengamos el montaje completado, se conectará el cable USB entre un puerto del PC y el conector de la placa Arduino UNO, siendo ya posible la comunicación de los módulos XBee con el ordenador y su configuración por medio del programa X-CTU como se verá en el apartado 3.2.3.1.

3.2.2.2. Cargar Sketch en la Placa Arduino UNO

Cuando se tenga que cargar un programa en la placa ArduinoUNO podremos hacerlo de dos maneras diferentes:

- Sin tener ninguna shield montada en la placa ArduinoUNO, se conectará ésta a un puerto del ordenador vía USB y la comunicación ya podrá realizarse sin problema alguno.
- En cambio, si se encuentra la XBee Shield montada en la placa ArduinoUNO, habrá que prestar especial atención a los jumpers, los cuales tendrán que estar en la posición USB. Recordar que ya no se deben conectar los pines RESET y GND, puesto que la comunicación que se quiere conseguir es ordenador-microcontrolador y viceversa.

3.2.2.3. Comunicación Inalámbrica entre Nodos

Para poder poner en funcionamiento una comunicación inalámbrica entre nodos se deberá:

- En primer lugar configurar los módulos XBee de radiofrecuencia tal cual es explicado en el apartado 3.2.2.1.
- Se cargará el sketch necesario en cada uno de los nodos como se indica en el apartado 3.2.2.2.
- A continuación, se hace el montaje de la placa ArduinoUNO + XBee Shield + módulos XBee. Ahora los jumpers de la XBee Shield deberán estar en la posición XBEE para que sea posible la comunicación inalámbrica entre nodos. Este montaje se aplicará en los dispositivos finales, como se aprecia en la Figura 46.

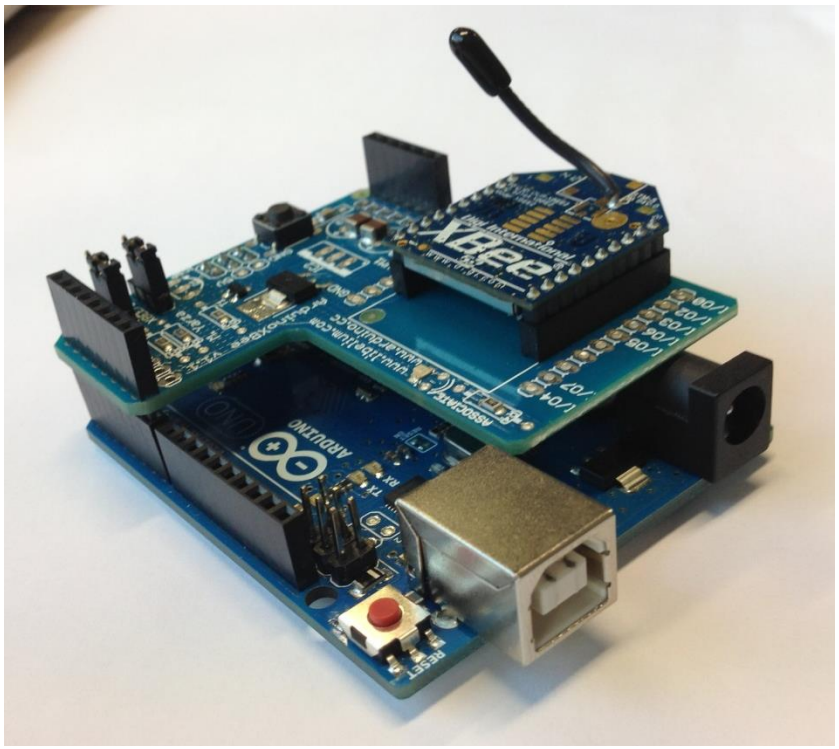


Figura 46. Dispositivo Final + XBee Shield + Módulo XBee

- En el caso del Coordinador se realizará el montaje de manera diferente al de los dispositivos finales. Primero habrá que conectar los cables al módulo XBee como se aprecia en la Figura 48 con ayuda del mapa de pines de la figura:

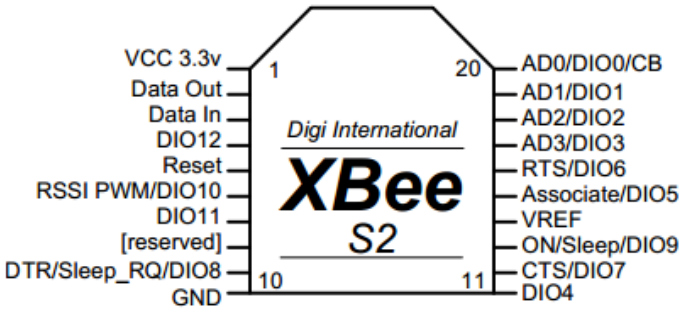


Figura 47. Mapa de pines del módulo XBee S2.

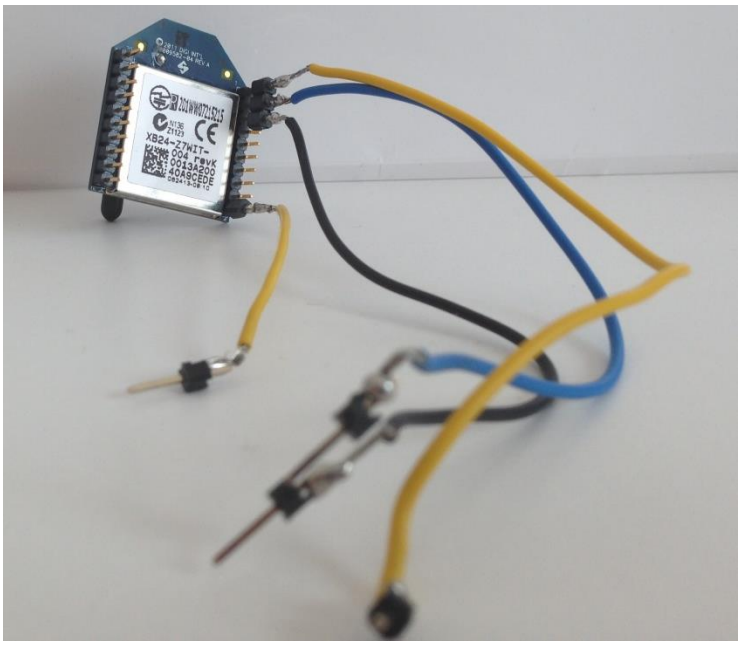


Figura 48. Conexión a pines módulo XBee.

Y por último, conectar el módulo XBee a la placa Arduino UNO (rev3) como en la Figura 49 con ayuda de la Tabla 2.

Módulo XBee	Arduino UNO (rev3)
VCC o 3.3V	3V3
TX o DOUT	Pin digital 2
RX o DIN	Pin digital 3
GND	GND

Tabla 2. Conexión módulo XBee a placa Arduino UNO (rev3).

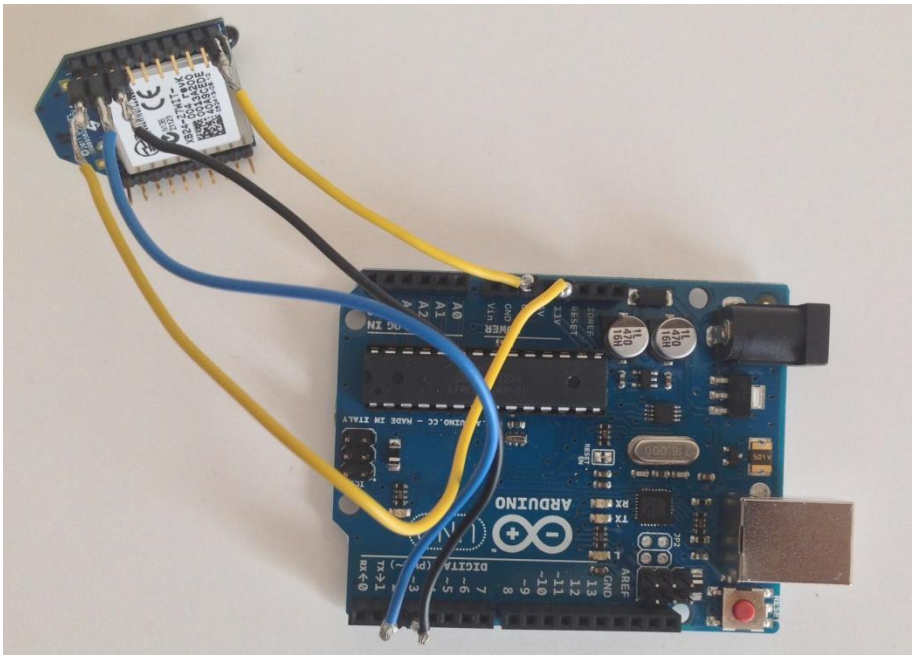


Figura 49. Coordinador + Módulo XBee

- Podremos alimentar el nodo bien vía USB, con un adaptador AC-DC (voltaje recomendado 7V-12V) o batería.

Figura 50. Alimentador eléctrico 12V 1A

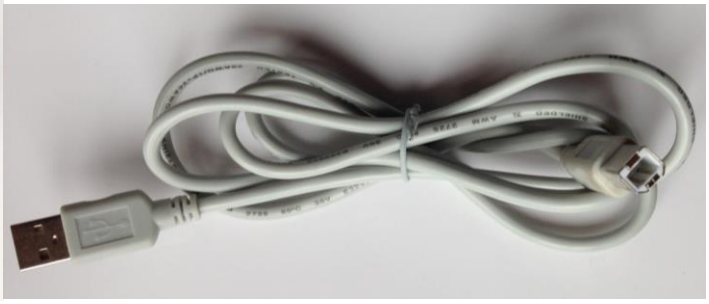


Figura 51. Cable USB tipo A-B

3.2.3. Configuraciones Software

3.2.3.1. Configuración Módulos XBee de Radiofrecuencia

A lo largo de este proyecto, el software utilizado para la configuración de los módulos XBee será el programa X-CTU proporcionado por Digi International, Inc., aunque hay disponibles otras opciones utilizando un software terminal (p.ej. PuTTY, HyperTerminal).

Hay que tener en cuenta que solo el programa X-CTU permite actualizar y cargar el firmware.

Una vez realizado el montaje del hardware, tal cual se explicó en el apartado 3.2.2.1, se podrá comenzar con la configuración de los módulos Xbee. Se tendrá que iniciar el programa X-CTU, el cual deberá mostrar la conexión USB como uno de los puertos disponibles. En Windows serán listados como uno de los puertos COM como se puede apreciar en la Figura 52. [**67**]

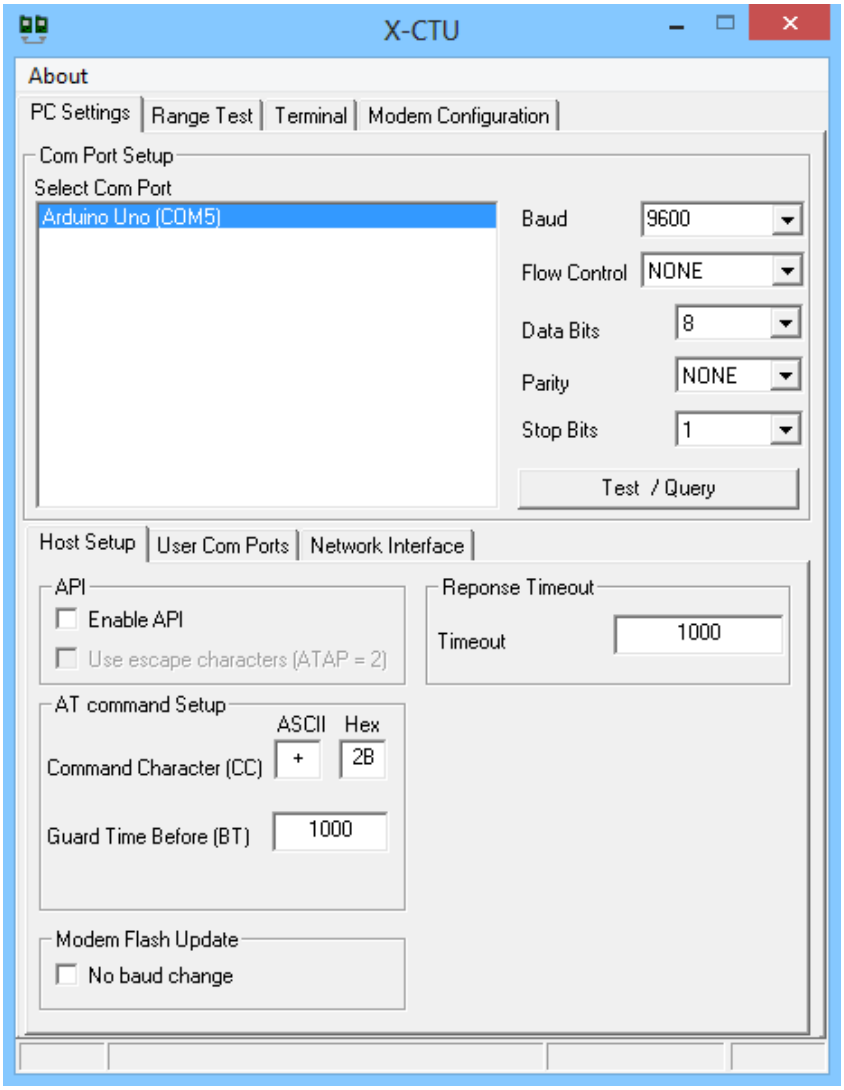


Figura 52. X-CTU Pantalla inicial

Se seleccionará el puerto apropiado de la ventana "Select Com Port" y aquí también, si se desea, se podrá modificar el campo "Baud Rate" que por defecto viene configurado en 9600 baudios. Además, tanto en el caso del coordinador como del dispositivo final, se seleccionará:

- En la ventana “PC Settings”, la opción “Enable API” y “Use escape characters (ATAP=2)”, como se muestra en la Figura 53.
- Y en la ventana “Modem Configuration”, la opción “AP- API Enable” a 2, como se puede ver en la Figura 54.

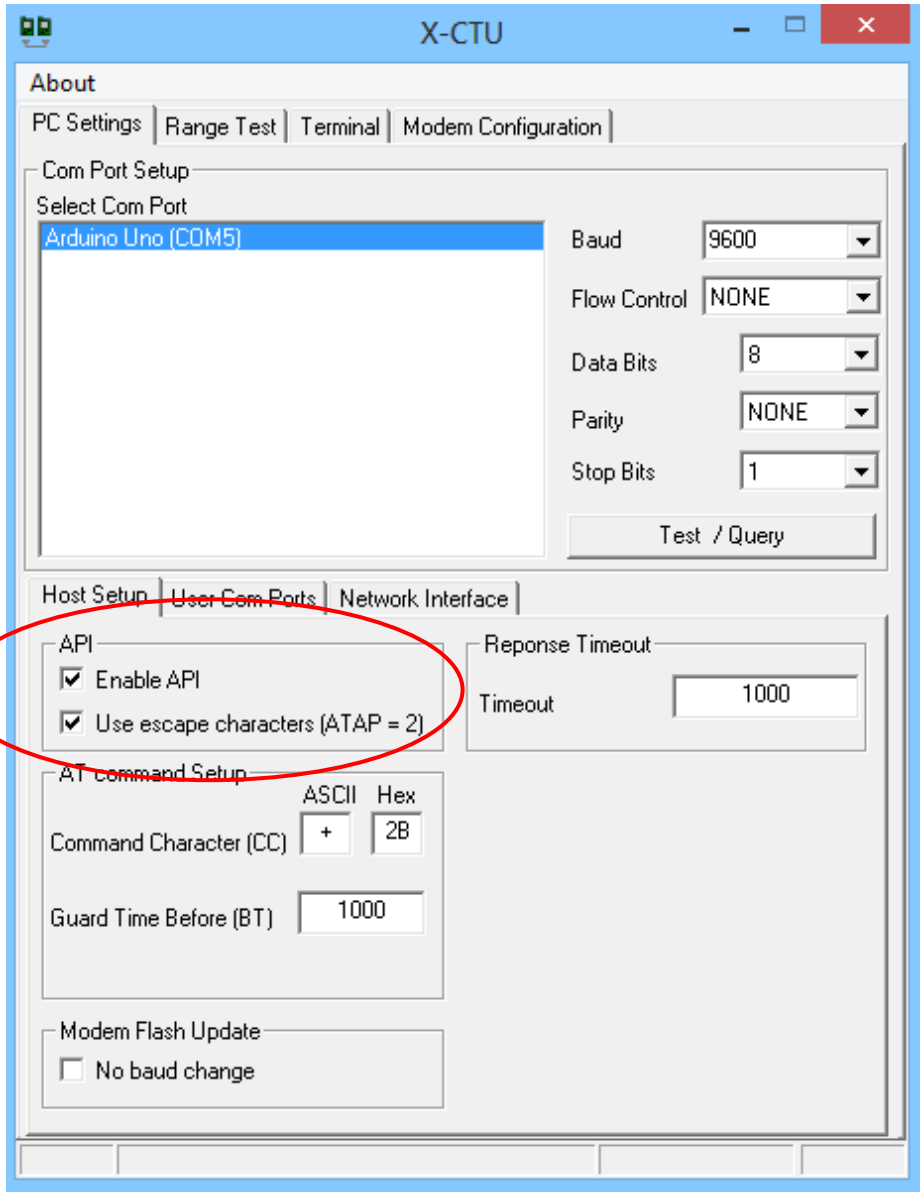


Figura 53. X-CTU Pantalla Inicial Coordinador

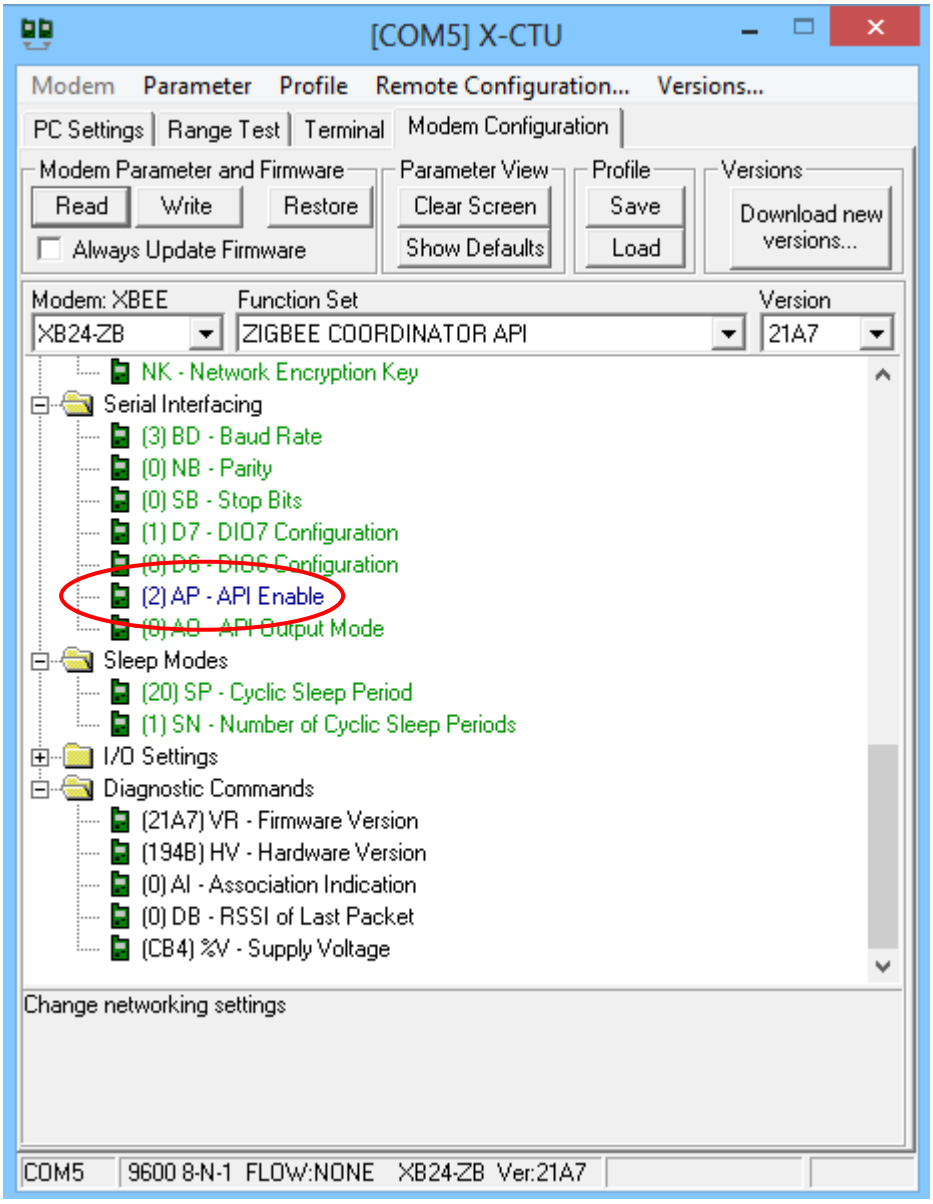


Figura 54. X-CTU Modem Configuration. Modo API

A continuación, hacer clic en el botón “Test/Query” el cual detecta la comunicación con el modem por el puerto serial-USB, en este caso COM5, para confirmar que la configuración por defecto en X-CTU funciona correctamente. Si la comunicación entre X-CTU y las radios XBee ha ido bien, mostrará un mensaje confirmando que la comunicación con el modem está bien, así como el tipo de modem y la versión del firmware. [**67**]

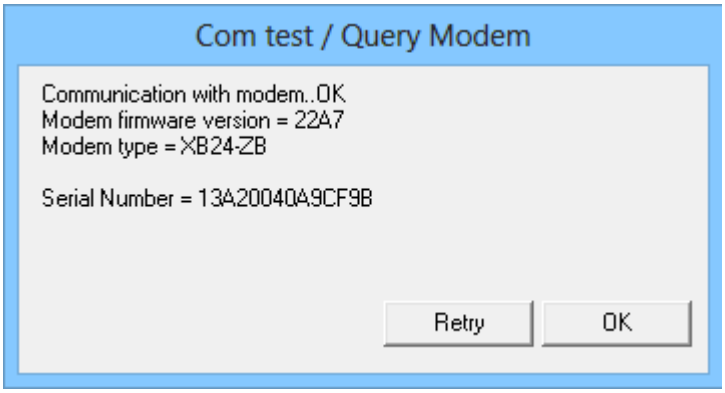


Figura 55. X-CTU Test de Confirmación

Ahora que ya se han probado los módulos XBee para la configuración básica, se debe prestar atención a que firmware están soportando y como están actualmente configurados. Para ello, hay que seleccionar la pestaña "Modem Configuration" y después hacer clic en el botón "Read". Esto muestra una pantalla con la actual configuración de la radio como se muestra en la Figura 56. [**67**]

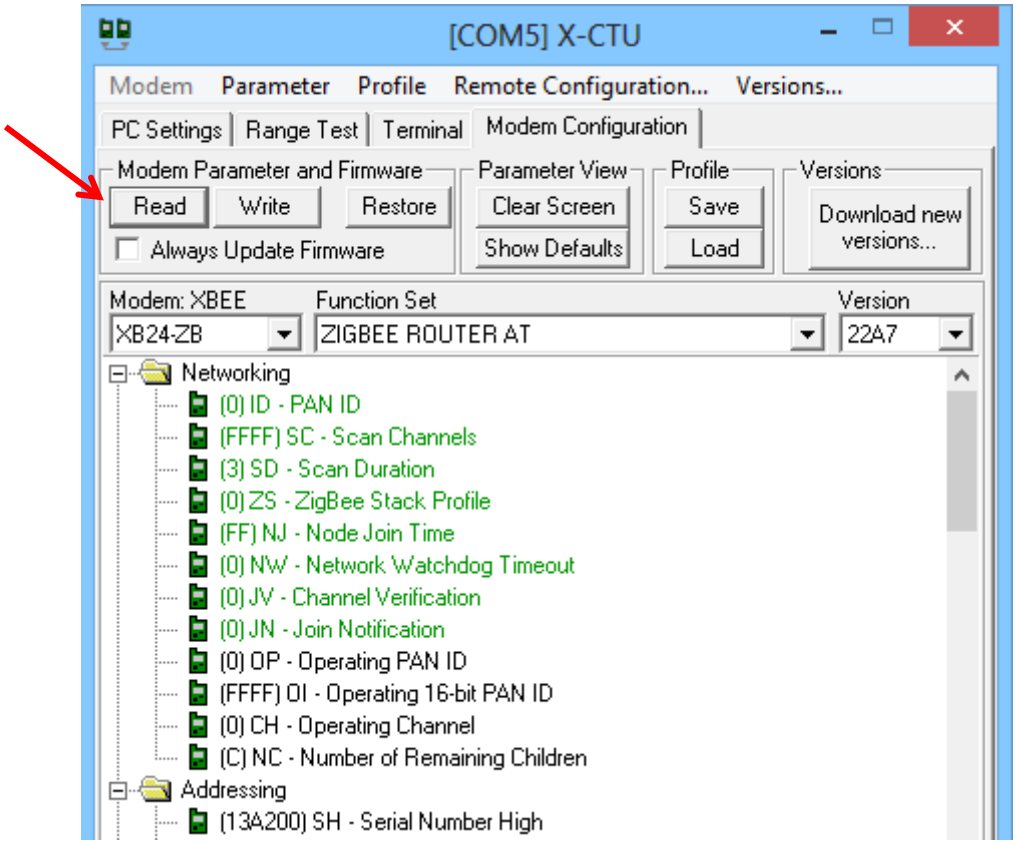


Figura 56. X-CTU Configuración Actual

Comenzamos la configuración definiendo los parámetros específicos para cada uno de los nodos de nuestra red:

- Seleccionar del desplegable "Modem XBEE" el tipo de radio del que se trata. A lo largo de este proyecto se trabajará con la radio XBee Serie 2, modelo XB24-Z7WIT-004. Por lo tanto, en este caso se ha de seleccionar la opción XB24-ZB.
- A continuación se debe definir qué tipo de nodo será el módulo XBee en cuestión: "Coordinator", "Router" o "End Device". En el desplegable "Function Set" se verá una lista de los diferentes firmwares que se pueden cargar. Además, se deberá tener en cuenta el modo de comunicación: AT o API.
- En general, se debe utilizar la versión de mayor número (hexadecimal) que aparece en el desplegable "Version".

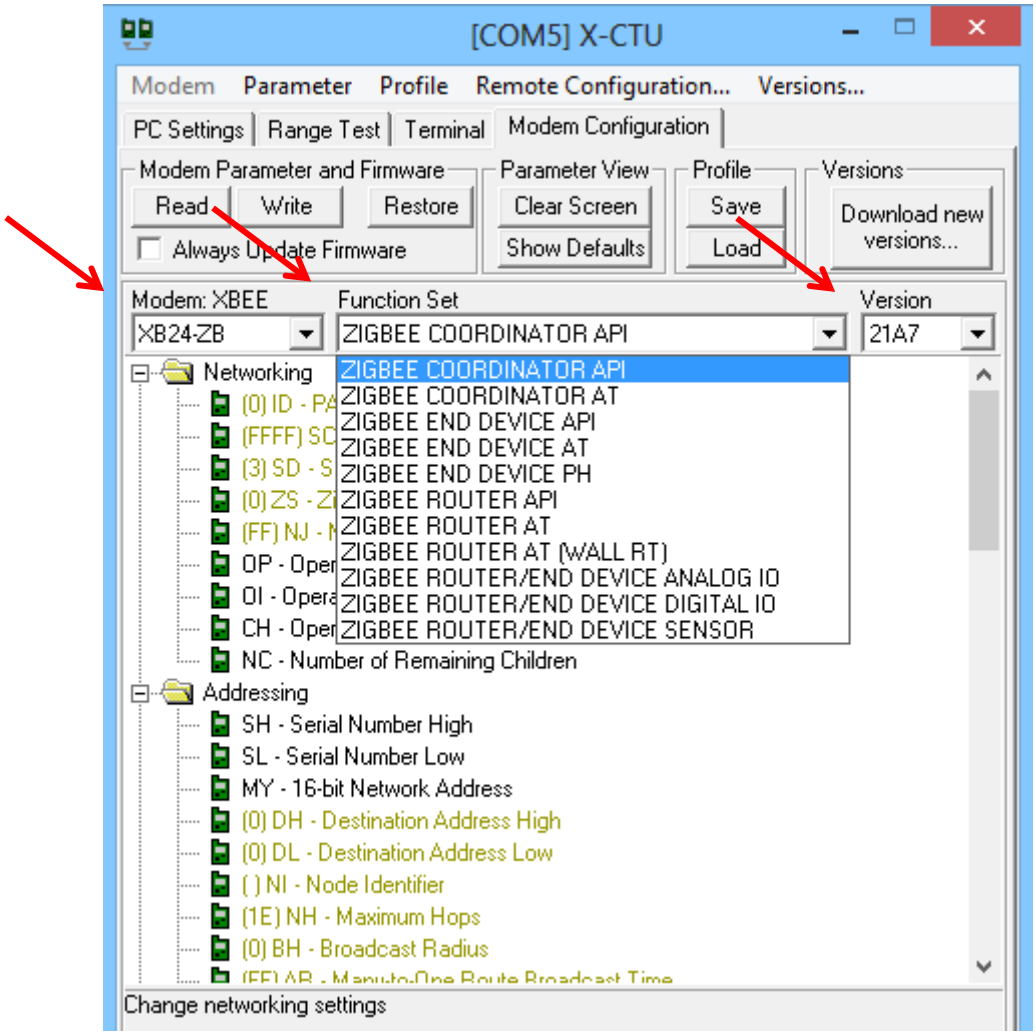


Figura 57. X-CTU Function Set & Version

- Además, en el campo "Networking" se deberá establecer un PAN ID (Personal Area Network Identifier) que tendrá que ser igual para todos los nodos de la misma red. Cada red ZigBee o PAN (Personal Area Network) está definida con un identificador único, PAN ID. Será un número con un rango válido de 0-0xFFFFFFFFFFFF o también puede ponerse ID=0 en el coordinador para que el PAN ID sea elegido aleatoriamente.

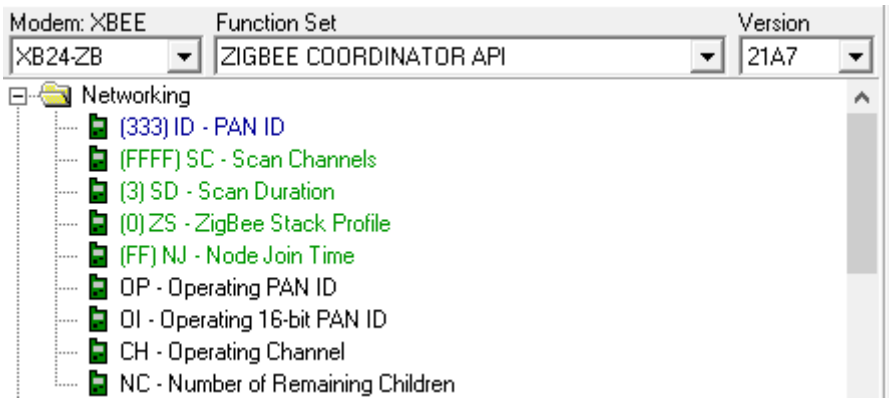


Figura 58. X-CTU PAN ID

- En este punto el coordinador tendría la configuración básica para poder comunicarse, como se muestra en la Figura 59. Para que el dispositivo final pueda comunicarse, primero tiene que formar parte de la red y para ello tiene que unirse a un coordinador o a un router y que actué de padre. Entonces le será asignada una dirección de 16-bit o dirección de red.

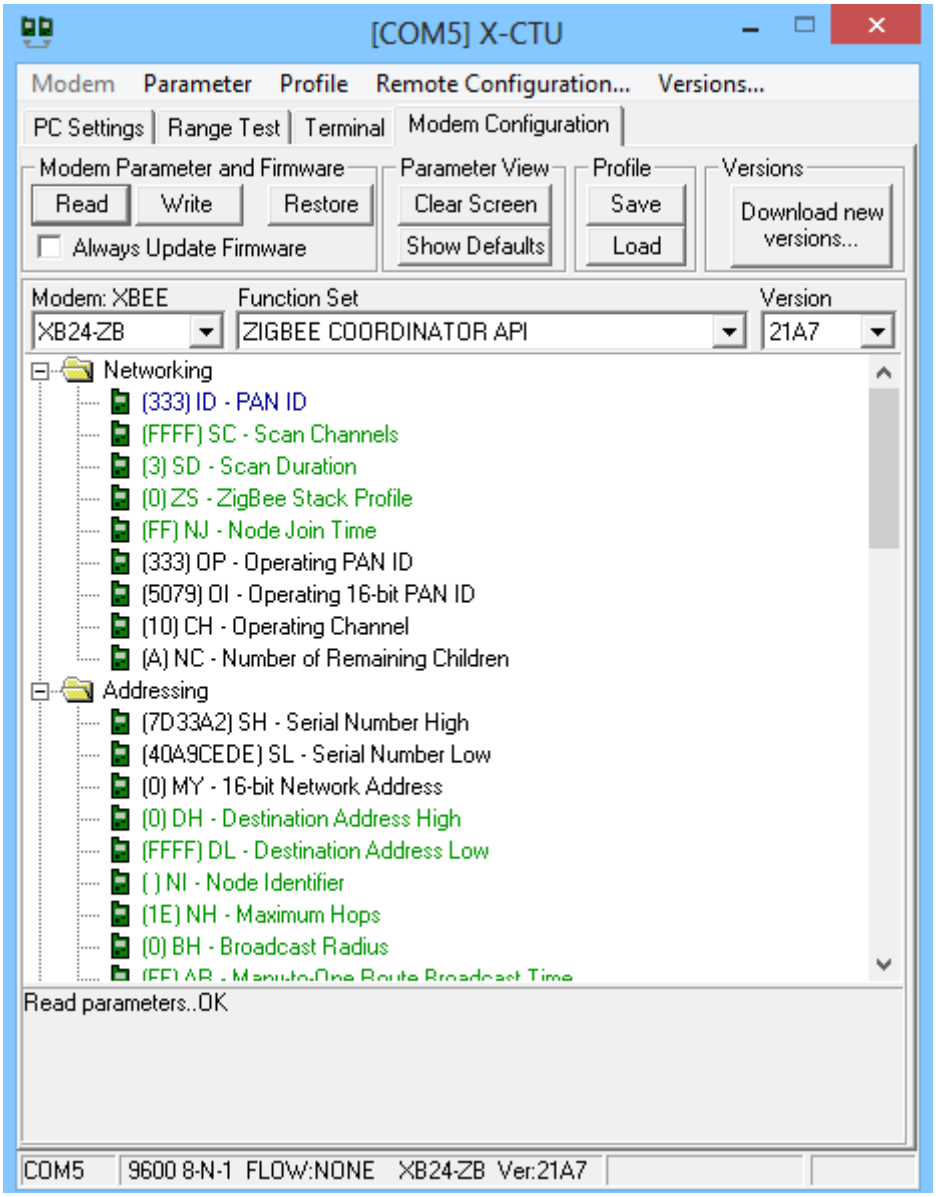


Figura 59. X-CTU Configuración Coordinador

- Para el dispositivo final y el router las direcciones destino (DH y DL) se tienen que especificar en el caso de que se quiera enviar información a un dispositivo concreto que no sea el coordinador, en cuyo caso se utiliza el valor por defecto DH=0 y DL=0, y serán los SH y SL del nodo destino.

  Teniendo en cuenta la decisión tomada en el apartado 3.2.3.2 en cuanto a la configuración del modo sleep en los dispositivos finales, se tendrá que modificar la opción "Sleep Mode" a 1, como se muestra en la Figura 61. En este proyecto se trabajará con dos dispositivos finales y su configuración final debe ser la que se muestra a continuación.

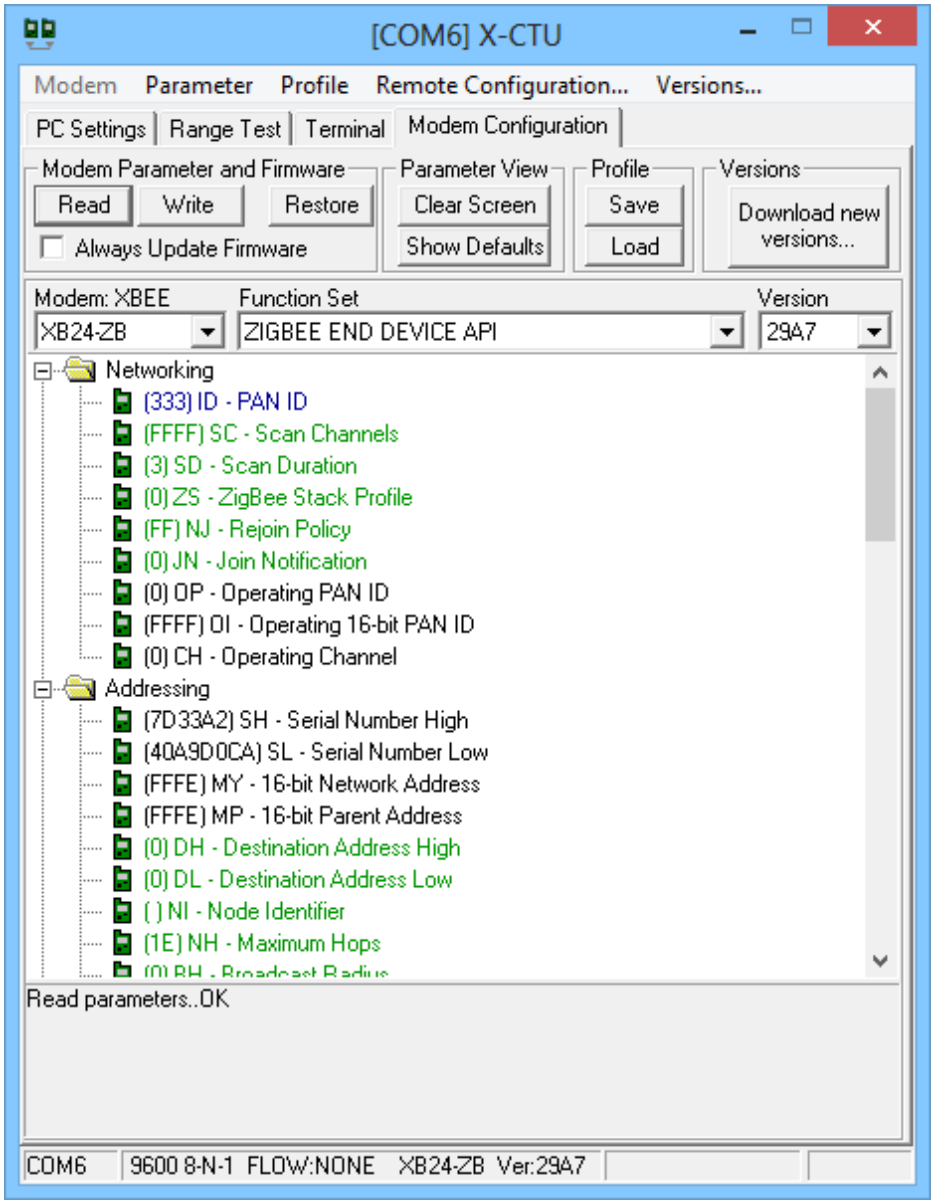


Figura 60. X-CTU Configuración Dispositivo Final

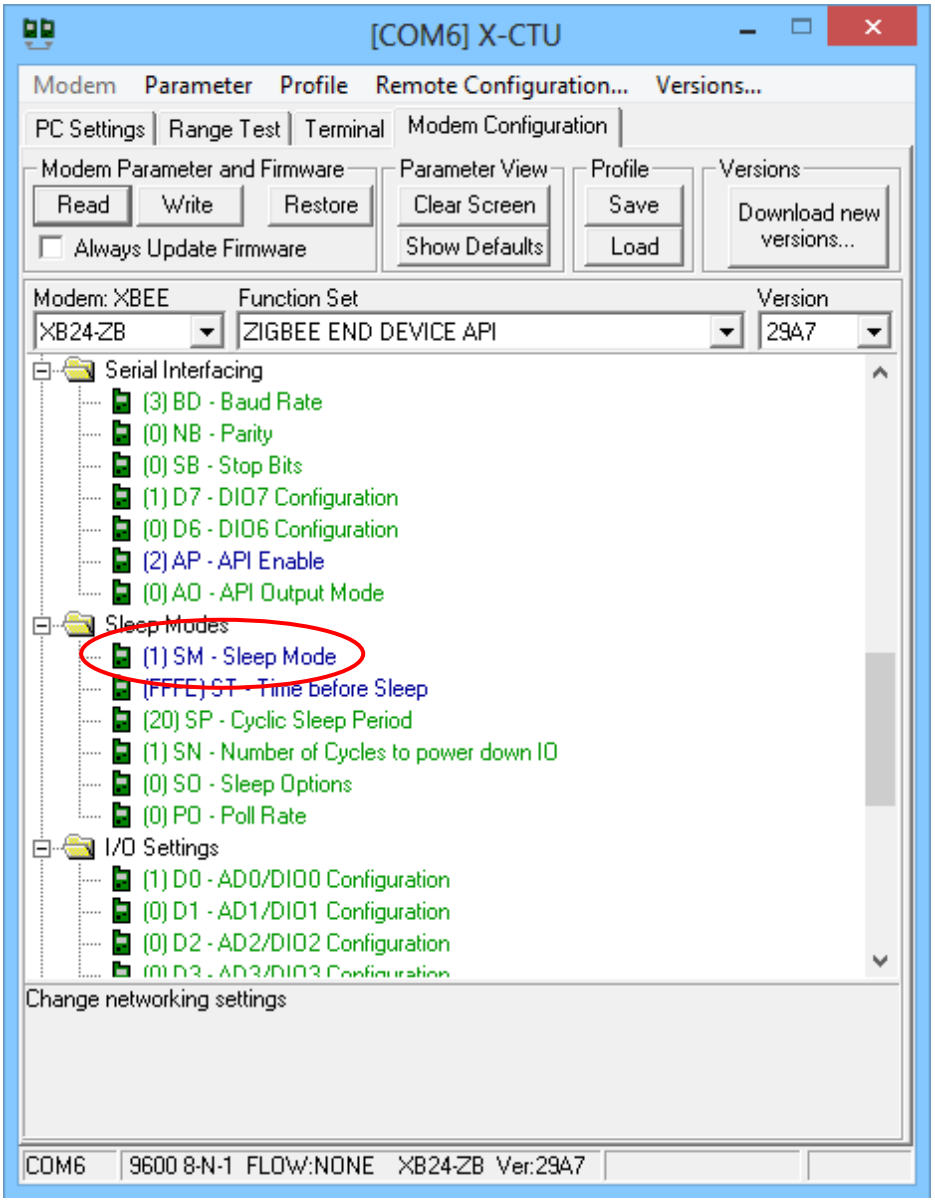


Figura 61. X-CTU Configuración Dispositivo Final

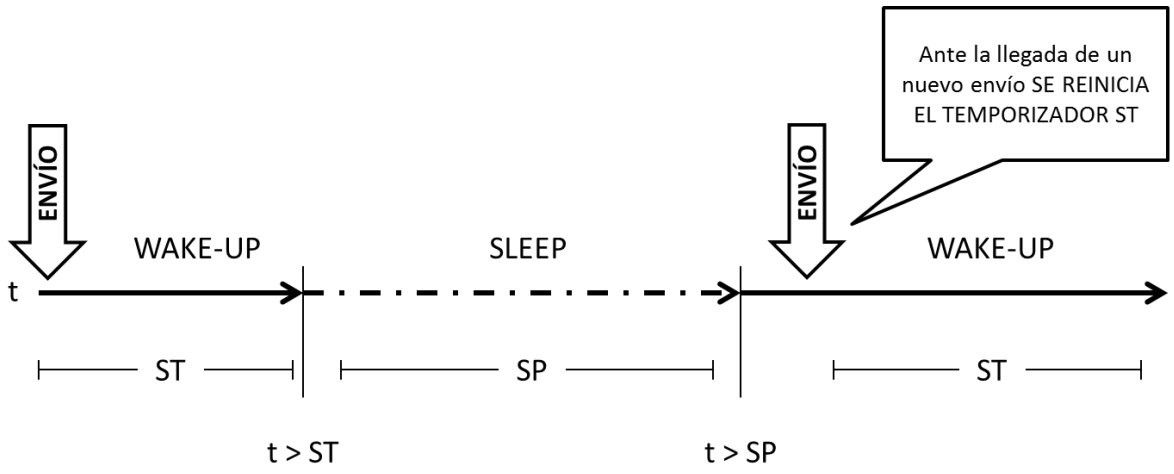
- El resto de parámetros pueden ser establecidos usando "Modem Configuration", la ventana Terminal y haciendo uso del modo API.
- Tanto para cargar el firmware elegido, como para guardar y escribir los parámetros en la radio se debe hacer clic en el botón "Write", si utilizamos "Modem Configuration".

Cuando finalice, el módulo estará preparado para comunicar. Ahora solo quedará cargar en cada uno de los nodos el sketch desarrollado como se explicó en el apartado 3.2.2.2.

3.2.3.2. Configuración Modo Sleep de los Módulos XBee

Se configurarán los módulos XBee con el modo "Sleep" utilizando el programa X-CTU. Para ello habrá que modificar una serie de parámetros:

- **SM (Sleep Mode)**: define el modo de "Sleep" en el módulo de radiofrecuencia. Pudiendo ser:
  - ✓ 0- Sleep disabled: un dispositivo final no puede ser configurado en este modo, solamente podrá trabajar en modo 1 y 4. Esto es porque el dispositivo final está diseñado para ahorrar batería.
  - ✓ 1- Pin sleep enabled. Se hará referencia más adelante.
  - ✓ 4- Cyclic sleep enabled. Se hará referencia más adelante.
  - ✓ 5- Cyclic sleep, pin wake.
- **SN (Number of Sleep periods)**: utiliza para extender el periodo de sueño (SP) actuando como un multiplicador de éste.
- **SP (Sleep Period)**: determina el tiempo que el dispositivo se encuentra dormido. El valor máximo es de 28 segundos y el mínimo de 320 milisegundos.
- **ST (Time before Sleep)**: es el tiempo que el módulo XBee está inactivo, sin enviar ni recibir datos, antes de pasar ha estado dormido. El valor máximo es de 65 segundos y el mínimo es de 1 milisegundo.
- **SO (Sleep Options)**: opciones del modo sleep:
  - ✓ 0x02: siempre despierto para un tiempo ST.
  - ✓ 0x04: el dispositivo se encontrará en estado dormido durante un tiempo SN*SP.



Los dispositivos finales soportan dos modos de funcionamiento, y estos, son de "sueño" (sleep modes):

- **Pin Sleep (SM=1)**: permite a un microcontrolador externo determinar cuando el Xbee se tiene que dormir y cuando despertar a través del pin Sleep_RQ (pin 9). Cuando este pin se activa (high) entonces completa todas las operaciones de recepción y envío que tiene pendientes, posteriormente entra en modo de bajo consumo. El XBee se despertará cuando el pin Sleep_RQ sea desactivado (low).
- **Cyclic Sleep (SM=4)**: permite dormir y despertar al dispositivo periódicamente. Es posible configurar los periodos de tiempo que se encuentra dormido y el tiempo que espera hasta que el Xbee se duerme a través de comandos AT (o utilizando XCTU).

Debido a que los dispositivos finales no pueden ser configurados con SM=0, que no es posible que un coordinador despierte al módulo XBee de un dispositivo final, y que en este proyecto no se busca que los dispositivos finales ahorren batería, ya que estarán conectados a la corriente eléctrica, se ha decidido utilizar el SM=1 (Pin Sleep). Para ello se soldará un cable al pin 9 del módulo XBee y éste conectarlo a pin GND de la placa Arduino, consiguiente así que este permanentemente despierto [**68**].

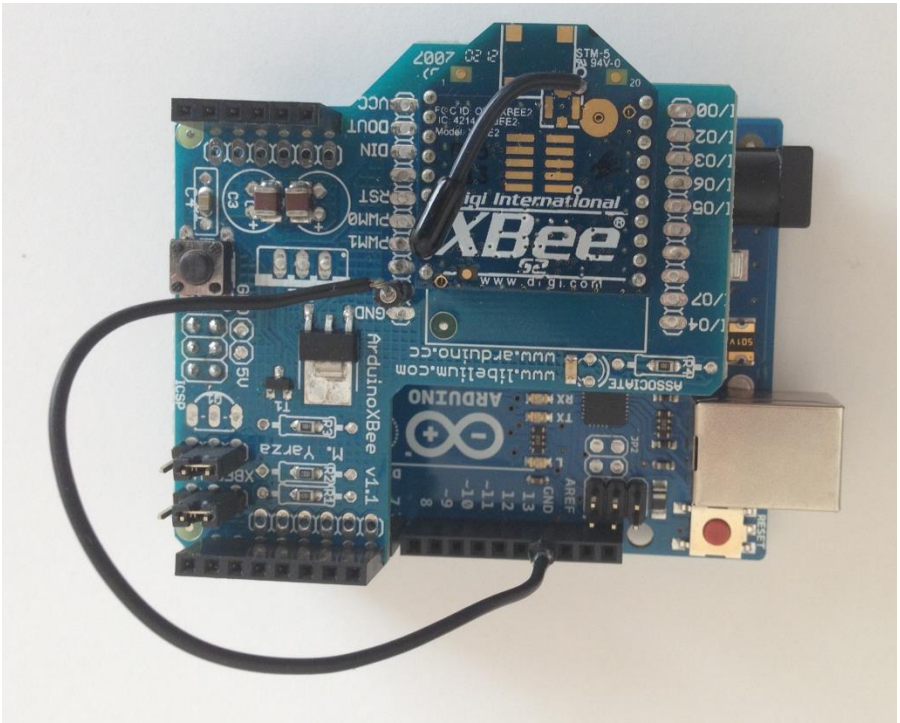


Figura 62. Configuración del pin sleep.

3.2.3.3. Configuración Librería RXTX

Como se expuso en el apartado 3.1.6.1, ante la problemática de la comunicación serie entre PC y Arduino se decidió utilizar en este proyecto la librería RXTX que ofrece soporte a multitud de plataformas hardware. Esta librería es nativa de Java que proporciona comunicación serie y paralelo para el kit de herramientas de desarrollo de Java (JDK). RXTX está disponible bajo la licencia GNU LGPL, así como las distribuciones binarias.

Para configuración de la librería RXTX se debe descargar el paquete correspondiente a la plataforma que se esté usando. En la página oficial de RXTX se pueden obtener estos paquetes:

http://rxtx.qbang.org/wiki/index.php/Download

Pero estos carecen de las versiones precompiladas de Windows x64, x86, ia64 (Intel Archictecture) y Linux x86, x86_64. Éstas son distribuidas por Mfizz Inc:

http://mfizz.com/oss/rxtx-for-java

Para este proyecto se descargó la versión, RXTX-2-2-20081207. Este paquete contiene un archivo de texto "Install.txt" que indica las instrucciones de instalación de la librería.

A continuación se explican las clases y métodos de la librería RXTX que se han utilizado en esta aplicación:

CommPortIdentifier. Es la clase central para controlar el acceso a las comunicaciones con los puertos y proporciona funcionalidades para descubrir puertos de comunicación, abrir puertos de comunicación, etc. Una aplicación utiliza primero los métodos definidos en *CommPortIdentifier* para negociar con el driver para descubrir que puertos para la comunicación están disponibles y luego seleccionará un puerto para la apertura. A continuación, utilizará métodos de otras clases como *CommPort*, *ParallelPort* y *SerialPort* para comunicarse a través del puerto.

CommPortIdentifier getPortIdentifiers (String portName)

- Obtiene un objeto de tipo Enumeration que contiene un objeto de

CommPort open (String appname, int timeout)

- Abre el puerto de comunicación y devuelve el objeto CommPort para el puerto.

CommPort. Es una clase abstracta que describe un puerto de comunicaciones. Incluye métodos de alto nivel para el control de E/S que son comunes a diferentes tipos de puertos de comunicaciones. *SerialPort* y *ParallelPort* son subclases de *CommPort* que incluyen métodos adicionales para el control de bajo nivel de los puertos de comunicaciones.

No hay constructores públicos para *CommPort*. Entonces una aplicación debe utilizar el método ***CommPortIdentifier.getPortIdentifiers*** para generar una lista de puertos disponibles. A continuación se elige un puerto de la lista y llama a ***CommPortIdentifier.open*** para crear un objeto *CommPort*. Por último, se hace un cast del objeto *CommPort* a una clase como *SerialPort* o *ParallelPort*. Después de que el puerto ha sido abierto se pueden configurar los parámetros con los métodos de las clases SerialPort y ParallelPort. Por último, se crearan flujos de datos de entrada y salida para escritura y lectura de datos.

void close ()

- Cierra las comunicaciones del puerto.

InputStream getInputStream()

- Devuelve un flujo de entrada.

OutPutStream getOutputStream()

- Devuelve un flujo de salida.

SerialPort. Define la funcionalidad mínima necesaria para puertos de comunicación serie.

void setSerialPortParams (int baudrate, int dataBits, int stopBits, int parity)

- Configura los parámetros del puerto.

void addEventListener (SerialPortEventListener lsnr)

- Registra al objeto SerialPorEventListener para que escuche SerialEvents.

void removeEventListener()

- Elimina el registro del detector de eventos registrado utilizando addEventListener.

void notifyOnDataAvailable (boolean enable)

- Expresa interés en recibir una notificación cuando los datos de entrada se encuentran disponibles.

3.2.3.4. Configuración Librería NewSoftSerial

Como se explicó en el apartado 3.1.6.2 para la comunicación serie entre PC y la placa Arduino UNO (rev3) se decidió utilizar los pines 0 (RX) y 1 (TX) que son los que incorpora la placa para dicha comunicación. El soporte nativo de serie pasa a través de una pieza de hardware (integrado en el chip) llamado UART. Este hardware permite al chip ATmega recibir la comunicación en serie incluso mientras se trabaja en otras tareas, siempre que haya espacio en el búfer de serie 64 bytes. Mientras que la comunicación entre coordinador y dispositivos finales se resolvió mediante la utilización de una librería, NewSoftSerial.

A partir de Arduino 1.0 y posteriores, la antigua librería SoftwareSerial [**69**] fue sustituida por NewSoftSerial, la cual es considerada actualmente como la oficial. De manera que a la hora de incluir la librería en el sketch se hará como en la Figura 63:

```
#include <SoftwareSerial.h>
```

Figura 63. Incluir librería NewSoftSerial en un sketch.

NewSoftSerial [**70**] es descendiente directo de AFSoftSerial de Ladyada, la cual introduce una recepción gestionada por interrupciones, lo que supuso una drástica mejora del rendimiento requerida por la librería nativa SoftwareSeial. Ha sido desarrollada para permitir la comunicación en serie en otros pines digitales del Arduino, utilizando el software para reproducir la funcionalidad. NewSoftSerial fue escrita sobre el principio de que se pueden tener tantos dispositivos conectados como la escasez de recursos lo permitan, siempre y cuando solo se utilice uno de ellos a la vez. La vinculación de la biblioteca NewSoftSerial a la aplicación añade aproximadamente 2000 bytes al tamaño.

En el proyecto se destinaron los pines digitales 2 y 3 para la recepción y envío de datos respectivamente a través del módulo XBee de radiofrecuencia. En la Figura 64 se puede ver la inicialización de la comunicación serie y SoftwareSerial.

```
/*  rxPin  2 pin usado para la recepción de datos
    provenientes del XBee (se conecta al DOUT del XBee)
    txPin  3 pin usado para el envío de datos
    al XBee (se conecta al DIN del XBee)*/
SoftwareSerial softSerial = SoftwareSerial(2,3);
/* crear el objeto de tipo XBee*/
XBee xbee = XBee();
void setup()
{
  /* inicializar comunicación serial a 9600 baudios:*/
  Serial.begin(9600);
  /* inicializar comunicación SoftwareSerial a 9600 baudios:*/
  softSerial.begin(9600);
  /* enlazar objeto XBee con la comunicación SoftwareSerial*/
  xbee.setSerial(softSerial);
}
```

Figura 64. Inicialización comunicación SoftwareSerial.

3.2.3.5. Configuración Librería XBee

Los módulos XBee pueden utilizar su conexión serie local de dos maneras muy diferentes. Las radios configuradas para el modo API utilizan un formato de datos-envolvente, ideal para ordenadores que hablan entre ellos, pero no es fácilmente legible por humanos. Este modo especifica cómo se envían y reciben desde el módulo XBee mensajes "commands", "command response" y "module status" mediante un "UART Data Frame". Los XBees que están configurados para utilizar el modo AT están diseñados para una interacción humana más directa [**60**].

En este proyecto se decidió utilizar el modo API porque [**71**]:

- Un nodo puede recibir datos de uno o más nodos remotos, siendo éste el Coordinador.
- Permite obtener la dirección origen a partir de los paquetes recibidos.
- Permite la transmisión de datos a múltiples destinos, sin entrar en el modo comando (modo AT para configuración de la radio a través de comandos AT).
- Ofrece la posibilidad de recibir el estado de éxito o fracaso de casa paquete de radiofrecuencia transmitido.
- Facilita la configuración de una radio remota mediante las características remotas del modo AT.
- Permite recibir paquetes de datos desde múltiples dispositivos y pudiendo identificar el dispositivo que realizó el envío.

- Además ofrece una API que facilita el envío de los datos.

La librería XBee de Arduino permite comunicarse con XBees en modo API, con soporte tanto para la Serie 1 (802.15.4) y la Serie 2 (ZB Pro / ZNet). El modo API se habilita cargando el firmware API siendo necesario que el parámetro AP este configurado a 2 (bytes de escape), ya que esta configuración ofrece la mejor fiabilidad.

Para la configuración de la librería se ha de descargar la versión más actualizada del paquete. Durante la realización de este proyecto la versión utilizada fue "xbee-arduino-0.4-softwareserial-beta" del siguiente enlace:

https://code.google.com/p/xbee-arduino/

Posteriormente se extrae el contenido del paquete (comprimido en formato RAR) y se copiará en Arduino\libraries.

Entre las tramas API existentes para el modo API, en el proyecto se utilizaron [**64**]:

- **ZigBee Transmit Request.** Una trama API de este tipo permite enviar datos como un paquete de RF al destino especificado. Se puede utilizar la dirección 0x000000000000FFFF para transmisión broadcast. Para enviar al coordinador se puede hacer de dos maneras: configurando la dirección de 64 bits toda a 0x00s y la de 16 bits a 0xFFFE, o configurando la dirección de 64 bits con la propia dirección de 64 bits del Coordinador y la dirección de 16 bits a 0x0000. Para el resto de las transmisiones, la dirección de 16 bits se configura con la dirección de 16 bits propia de cada dispositivo, ya que puede ayudar a mejorar el rendimiento cuando se transmiten a múltiples destinos. Si no se conoce la dirección de 16 bits, este campo se debe establecer en 0xFFFE (desconocido). La trama Transmit Status (0x8B) indicará el descubrimiento de la dirección de 16 bits, si tiene éxito.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x16	
	Frame-specific Data	Frame Type	3	0x10	
		Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x0A	
			11	0x01	
			LSB 12	0x27	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Broadcast Radius	15	0x00	Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value.
		Options	16	0x00	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries and route repair 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout Enabling APS encryption presumes the source and destination have been authenticated. It also decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). The extended transmission timeout is needed when addressing sleeping end devices.It also increases the retry interval between retries to compensate for end device polling.See Chapter 4, Transmission Timeouts, Extended Timeout for a description. Unused bits must be set to 0.
		RF Data	17	0x54	Data that is sent to the destination device
			18	0x78	
			19	0x44	
			20	0x61	
			21	0x74	
			22	0x61	
			23	0x30	
			24	0x41	
	Checksum		25	0x13	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figura 65. ZigBee Transmit Request.

- **ZigBee Transmit Status.** Cuando se completa el envío de un ZigBee Transmit Request se recibirá un paquete de mensaje de tipo ZigBee Transmit Status que indicará el estado de la entrega.
- **ZigBee Receive Packet.** Es el paquete que contiene información enviada desde un determinado dispositivo. El campo "received data" contiene los datos del paquete.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x11	
	Frame-specific Data	Frame Type	3	0x90	
		64-bit Source Address	MSB 4	0x00	
			5	0x13	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender
			LSB 13	0x84	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet 0x20 - Packet encrypted with APS encryption 0x40 - Packet was sent from an end device (if known) Note: Option values can be combined. For example, a 0x40 and a 0x01 will show as a 0x41. Other possible values 0x21, 0x22, 0x41, 0x42, 0x60, 0x61, 0x62.
		Received Data	15	0x52	Received RF data
			16	0x78	
			17	0x44	
			18	0x61	
			19	0x74	
			20	0x61	
	Checksum		21	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figura 66. ZigBee Receive Packet.

3.2.4. Desarrollos

A lo largo de este apartado se explicarán con mayor profundidad los distintos desarrollos llevados a cabo tanto las placas Arduino UNO (rev3) como en el PC.

3.2.4.1. Sketch para Monitorización de Temperatura.

Con este sketch se pretende monitorizar la temperatura en grados Celsius y Fahrenheit de un determinado entorno de manera remota. Además, el dispositivo final será capaz de, una vez establecida la conexión con el coordinador, identificarse informando de sus características y servicios ofrecidos.

El dispositivo final actuará como nodo sensor tomando medidas y enviándolas cuando el usuario, por medio de una petición, solicite dicha información.

El código de este dispositivo final se divide en tres partes diferenciadas:

- **Descubrimiento de dispositivo final.** Como se ha comentado anteriormente, los dispositivos finales en este sistema son capaces de identificarse indicando sus características y servicios ofrecidos. Esto sucederá una única vez mientras el dispositivo este encendido. Se producirá cuando éste se haya unido a la red y este seguro de que se han establecido una dirección de 16-bit y una ruta al nodo destino, en este caso, el coordinador.

  La publicación de las características y los servicios se hace utilizando el formato JSON.

```
String json=
"{\"deviceType\":\"temperature\",\"deviceId\":\"0013a20040a9cf9b
\",\"parameters\":[{\"name\":\"outputType\",\"default\":\"json\"
},{\"name\":\"ignoreErrors\",\"optional\":true}],\"services\":[{
\"name\":\"getTemperature\",\"parameters\":[{\"type\":\"char\"}]
,\"returns\":\"float\"}]}";
```

Figura 67. Publicación de las características y servicios de un dispositivo final en formato JSON.

  Para llevar a cabo esta publicación se ha de fragmentar, tal y como se expuso en el apartado 3.1.6.3. Para lo cual, se ha implementado un código que permita la fragmentación de este mensaje y reenvió de los paquetes hasta que se haya establecido la conexión.

  Puesto que el envío solo va a realizarse una única vez mientras el dispositivo este encendido, el código de fragmentación y envío se situará en la función *setup()*, la cual se ejecuta al comenzar el sketch una sola vez.

La fragmentación se realizará de acuerdo al protocolo planteado en el apartado 3.1.7, donde quedaban 79 bytes libres para carga útil. Por lo tanto, se fragmentará el mensaje de la Figura 67 en fragmentos con una longitud de 79 bytes como máximo. Habrá que prestar especial atención al último fragmento, pues habrá que indicar en la PDU que se trata del último paquete perteneciente a ese mensaje que se envía.

Debido a que en ocasiones el envío de los paquetes ZigBee Transmit Request se realiza antes de que el módulo XBee del dispositivo final se haya unido a la red, se comprueba que los paquetes han llegado al destino, el coordinador, como se aprecia en la Figura 68. Cuando se completa el envío de un ZigBee Transmit Request se recibirá un mensaje tipo ZigBee Transmit Status que indicará el estado de la entrega en el campo "Delivery Status", en este caso, se espera que sea "SUCCESS"

```
/* Después de enviar el paquete TX Request se espera una
respuesta de tipo Status Response.
Se espera hasta medio segundo a recibir el paquete*/
if (xbee.readPacket(500)){
   if (xbee.getResponse().getApiId()== ZB_TX_STATUS_RESPONSE){
       xbee.getResponse().getZBTxStatusResponse(txStatus);
      /* obtener el byte delivery status, se espera que sea
      SUCCESS para confirmar que se ha enviado correctamente*/
       if (txStatus.getDeliveryStatus() == SUCCESS){
          // El paquete se ha enviado correctamente.
       }else{
          i--;
          /* Error. Paquete no recibido.
             Se reenvía el paquete.*/
       }
   }
}
```

Figura 68. Reenvío de paquetes durante el proceso de fragmentación.

- **Recepción de peticiones.** Se esperan recibir peticiones realizas por los usuarios en cualquier momento pudiendo información, en este caso, acerca de la temperatura del entorno en el que se encuentra el dispositivo final.

  Debido a esto se estará leyendo continuamente esperando la llegada de paquetes de tipo ZigBee Receive Packet, los cuales contienen información enviada desde un determinado dispositivo, en este caso, el coordinador. Por lo tanto, esta parte del sketch se implementará en la función *loop()*, la cual se ejecuta repetitivamente.

```
xbee.readPacket();
  if (xbee.getResponse().isAvailable()) {
    if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
      /* ver de qué tipo de paquete se trata */
      xbee.getResponse().getZBRxResponse(rx);
      /* ..... Resto de código para procesado paquete.....  */
      }
    }
  }
}
```

Figura 69. Recepción de peticiones.

Una vez recibido el mensaje, se procederá a analizar la PDU obteniendo el "tipo de payload" que deberá ser 1 (REQUEST) y la carga útil que podrá ser "C" en el caso de temperatura en Celsius o "F" en Fahrenheit. En función de la carga útil recibida, se procesará el valor medido por el sensor de temperatura como Celsius o Fahrenheit, obteniendo así el resultado esperado por el usuario.

- **Envío de respuestas.** En último lugar habrá que crear una PDU de respuesta al usuario con el valor obtenido. Esta parte de código se implementará en la función *loop()* al igual que la petición la recepción de peticiones.

Circuito y conexiones con Arduino UNO (rev3).

- ✓ Resistencia: 2000Ω.
- ✓ Sensor de temperatura: LM335Z

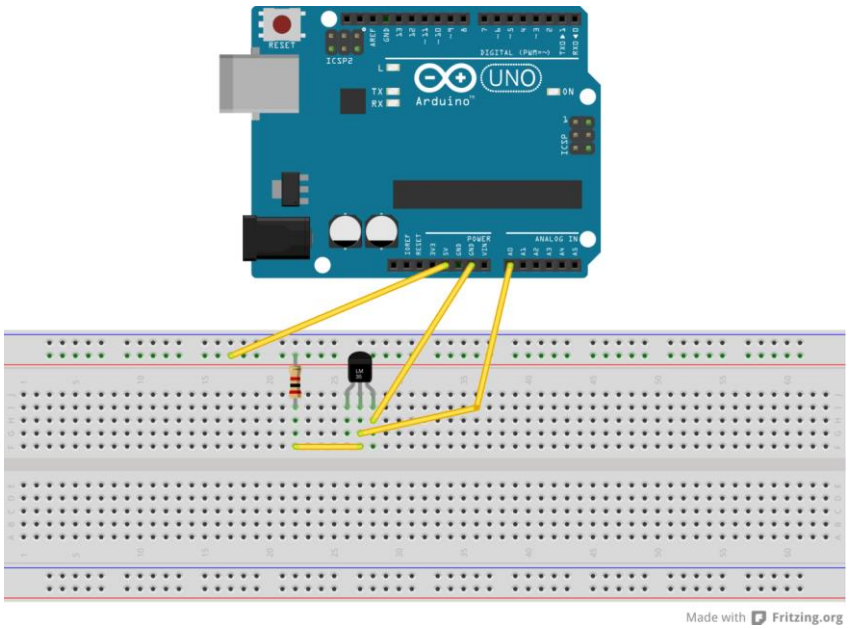


Figura 70. Montaje Arduino con Circuito Sensor de Temperatura.

3.2.4.2. Sketch para Monitorización de LED

Con este sketch se pretende actuar sobre un LED de manera remota. Además, el dispositivo final será capaz de, una vez establecida la conexión con el coordinador, identificarse informando de sus características y servicios ofrecidos.

El dispositivo final actuará de nodo actuador, encendiendo o apagando un LED, en función de lo deseado por el usuario.

El código de este dispositivo final se divide en dos partes diferenciadas:

- **Descubrimiento de dispositivo final.** El procedimiento de descubrimiento es idéntico al explicado en el apartado 3.2.4.1 para el dispositivo final con sensor de temperatura. La única diferencia se encuentra en el mensaje con formato JSON, como se muestra en la Figura 71.

```
String json=
"{\"deviceType\":\"led\",\"deviceId\":\"0013a20040a9d0ca\",\"par
ameters\":[{\"name\":\"outputType\",\"default\":\"json\"},{\"nam
e\":\"ignoreErrors\",\"optional\":true}],\"services\":[{\"name\"
:\"switch\",\"parameters\":[{\"type\":\"boolean\"}],\"returns\":
\"void\"}]}";
```

Figura 71. Publicación de las características y servicios de un dispositivo final en formato JSON.

- **Recepción de peticiones.** Se esperan recibir peticiones realizas por los usuarios en cualquier momento pudiendo actuar sobre un LED conectado al dispositivo final.

  Como en el caso del apartado 3.2.4.1 se estará leyendo continuamente esperando la llegada de paquetes de tipo ZigBee Receive Packet y por lo tanto está parte del sketch se implementará en la función *loop ()*.

```
xbee.readPacket();
  if (xbee.getResponse().isAvailable()) {
    if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
      /* ver de qué tipo de paquete se trata */
      xbee.getResponse().getZBRxResponse(rx);
      /* ……. Resto de código para procesado paquete…… */
      }
    }
  }
}
```

Figura 72. Recepción de ZigBee Receive Packets y procesado de éstos.

Una vez recibido el mensaje, se procederá a analizar la PDU obteniendo el "tipo de payload" que deberá ser 1 (REQUEST) y la carga útil que podrá ser "1" en

el caso de querer encender el LED o "0" si lo que se quiere es apagarlo. En función de la carga útil recibida, se actuará sobre dicho LED teniendo en cuenta el estado actual del LED.

Circuito y conexiones con Arduino UNO (rev3).

- ✓ Resistencia: 220 Ω.
- ✓ LED

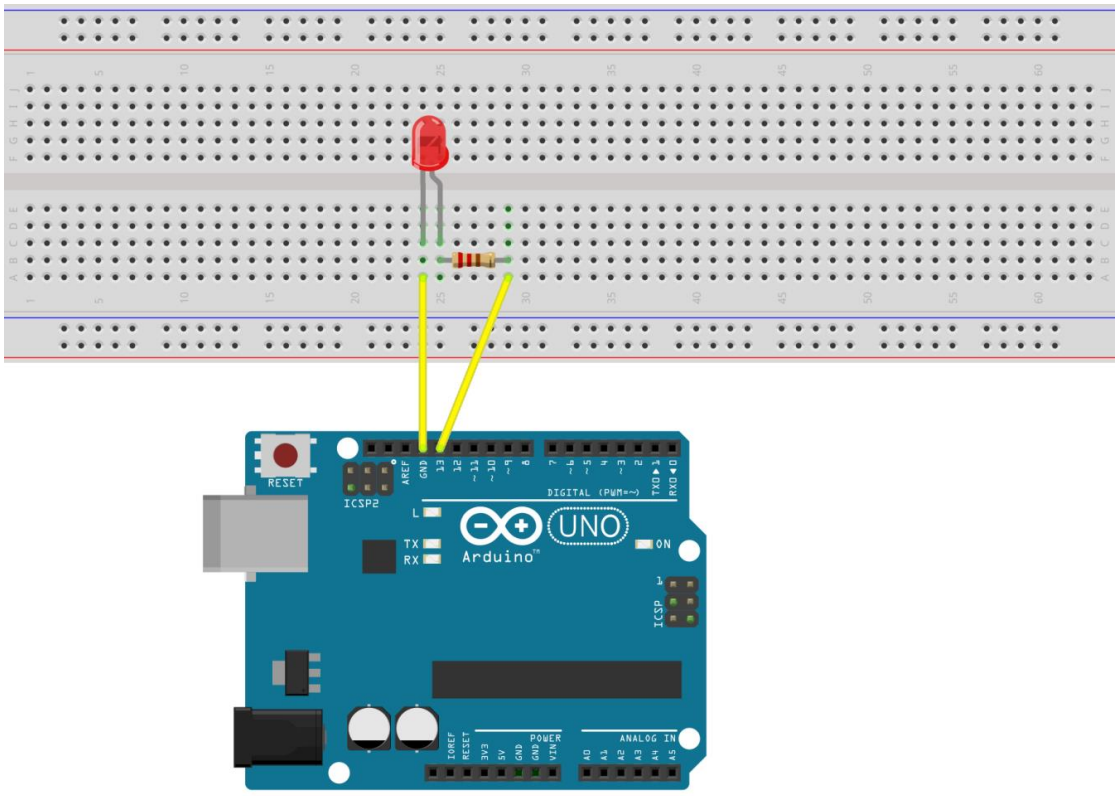


Figura 73. Montaje Arduino con Circuito LED.

3.2.4.3. Sketch para Coordinador

El dispositivo coordinador actuará como gateway entre la red de sensores y actuadores y el PC, elemento que funcionará como estación base. Por consiguiente este sketch tendrá que ser capaz de recibir peticiones por parte del PC reenviándolas al correspondiente dispositivo final y a su vez, procesar las respuestas recibidas por parte dichos dispositivos reenviándolas al PC en contestación a la petición.

Por lo que se puede dividir el código en dos partes, las cuales estarán implementadas en la función *loop ()* puesto que se han de estar ejecutando constantemente:

- **Recepción de peticiones por parte del PC y reenvió de éstas a los correspondientes dispositivos finales:** el coordinador estará continuamente comprobando si hay datos disponibles procedentes de la comunicación serie con el PC. Si es así, almacenará la dirección destino y la PDU de petición, pudiendo ya enviar dicha PDU al dispositivo final con esa dirección destino en un mensaje de tipo ZigBee Request Packet.

- **Recepción de respuestas por parte de los dispositivos finales y reenvió de éstas al PC:** en este caso el coordinador estará esperando una respuesta por parte de los dispositivos finales a los que haya enviado una petición. Está respuesta podrá ser de tipo "Status Response" que indicará si la petición se envió correctamente o de tipo "Response" la cual contendrá la respuesta a la petición.

  En el primer caso confirmaremos que el dispositivo final recibió la petición sin problemas y que está procesándola. En el segundo caso, si se trata de un mensaje de tipo ZigBee Response Packet, se obtendrá de éste la dirección origen para tener conocimiento de que dispositivo final proporciona la respuesta, como se en la y la PDU de respuesta.

```
/* Objeto reutilizable de respuesta de tipo Serie 2*/
ZBRxResponse rx = ZBRxResponse();
/* ............*/
/* Ver de qué tipo de paquete se trata */
if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE){
  xbee.getResponse().getZBRxResponse(rx);
  /* Obtener la dirección origen del paquete de respuesta*/
  XBeeAddress64 addressSource=rx.getRemoteAddress64();
  uint32_t msbDir=addressSource.getMsb();
  uint32_t lsbDir=addressSource.getLsb();
  /* ....... Resto de código de procesado .....*/
}
```

Figura 74. Recepción de Respuestas en el Coordinador.

  Por último se reenviará la PDU de respuesta al PC indicando la dirección del dispositivo final al que pertenece dicha respuesta.

También es de especial interés destacar que la inicialización de las comunicaciones inalámbricas en el código sufre ciertos cambios respeto a los dispositivos finales. Esto es debido a que la conexión hardware del módulo XBee es diferente a la realizada en los dispositivos finales como se explicó en el apartado 3.2.2.1. Y por ello se ha de incluir en el código del sketch la librería NewSoftSerial, que como se comentó en el 3.2.3.4 a partir de Arduino 1.0 está incluida en el núcleo de Arduino llamándose SoftwareSerial.

3.2.4.4. Aplicación Java

App. Se trata de la clase principal de la aplicación que se encarga de instanciar los objetos: *Communications*, *DeviceManager*, *HelloProcess*, *ResponseProcess*, *WMain* y *EventController*. Se almacenan dichas instancias en el objeto *EventController*, definido como una variable estática. Además se crea un hilo para la ejecución de la GUI y otro para la lógica de la aplicación.

EventController. Es una clase intermedia que almacena las instancias de los objetos que posteriormente será necesario utilizar en otras clases, teniendo así acceso a los métodos de éstos. Esto permite la separación de los datos y la lógica de la interfaz gráfica.

SerialCommunications. Clase que se encarga de las comunicaciones serie a través de la librería Java RXTX entre el puerto serie de la placa Arduino UNO (rev3) y el PC. Implementa la interfaz *SerialPortEventListener*.

void initialized ()

- Inicializa las comunicaciones con el puerto serie:
  - Se buscarán todos los puertos disponibles detectados por el PC, comparandolos con el que se desea trabajar.
  - Una vez obtenido dicho puerto, se procederá a su apertura.
  - Configuración de los parámetros del puerto serie (baud rate, data bits, stop bits, parity), que han de coincidir con los de la placa Arduino UNO (rev3).
  - También se abrirán los flujos de datos.
  - Por último, añadir los detectores (listeners) de eventos.

synchronized serialEvent (SerialPortEvent oEvent)

- Se encarga de manejar los eventos detectados en un puerto serie en particular. En este caso particular, se tratará el evento *SerialPortEvent.DATA_AVAILABLE* que informa de si hay datos disponibles en el puerto serie.
  - Se leerán todos los datos entrantes y se irán almacenando en un buffer hasta que se encuentre el indicador de final de paquete y no se trate de los 8 primeros bytes correspondientes a la XBeeAddress. Entonces se habrá recibido un paquete completo y será cuando se invoque al método *receive()* de la clase Communications pasandole como argumento un array de bytes que contiene dicho paquete.

synchronized void writeData (byte[ ] data)

- Permite escribir datos en el puerto serie.

synchronized void close()

- Debe ser llamado cuando se deje de usar el puerto. También se cerrarán los flujos de datos.

Communications. Clase que como su propio nombre indica se ocupa de las comunicaciones de la aplicación. Recibe y envía paquetes a través del puerto serie. Asigna los paquetes recibidos al suscritor encargado de su procesado y formar paquetes para su posterior envío. Implementa la interfaz *ICommunications.*

Communicactions()

- Constructor que inicializa las comunicaciones serie y crea la lista que relaciona suscriptores con tipo de eventos (HELLO_RECEIVED o RESPONSE_RECEIVED).

void subscribe (EventManger suscriber, int eventType)

- Se encarga de suscribir, en la lista que relaciona suscriptores con tipo de evento, cada suscriptor con el evento que le corresponde.

void send (String xbeeAddres, byte[] message, int longMessage, int messageType)

- Se ocupa del envío de paquetes hacia el puerto serie. Se le indica la dirección destino, la carga útil, la longitud de la carga útil y el tipo de mensaje (REQUEST). Con estos datos se formará el paquete y se enviará a través del puerto serie a la mota coordinador.

void receive (byte[ ] data)

- Se ocupa de enviar paquetes, recibidos por el puerto serie, al suscriptor que se encarga de procesarlos. En función del tipo de pdu (HELLO, RESPONSE) que se trate se buscará al suscriptor y éste procesará dicho paquete.

EventManager searchSuscriber (int event)

- Busca un suscriptor en función del evento pasado como argumento en la lista que almacena esta relación.

void closeSerial()

- Cierra las comunicaciones serie.

SuscriberEventRelationship. Clase que permite relacionar un determinado suscriptor con un cierto evento. Hasta el momento se ha trabajado con dos tipos de suscriptores (HelloProcess y ResponseProcess) y dos tipos de eventos (HELLO_RECEIVED, RESPONSE_RECEIVED).

SuscriberEventRelationship (EventManager eventManager, int event)

- Recibe como parámetro el suscriptor y el evento relacionados.

Métodos getter y setter

EventManager. Clase abstracta que opera como gestor de eventos. Será extendida por dos clases, *HelloProcess* y *ResponseProcess*, las cuales actúan como suscritores de los eventos que se pueden producir en la aplicación hasta el momento. Implementa la interfaz *ISuscriptor*.

void eventProcessing (int eventType, byte[] eventInformation)

- Método que se encarga de procesar el paquete recibido como un array de bytes.

HelloProcess. Clase que hereda de la clase *EventManager* y que actúa como suscriptor del evento HELLO_RECEIVED. Es la encargada de procesar los paquetes que contienen los fragmentos del mensaje JSON, publicado por los dispositivos finales, y reensamblarlos. Una vez esté formado el mensaje se llamará al método *addDevice ()* el cuál procesará dicho mensaje JSON y añadirá el dispositivo al conjunto de dispositivos finales.

HelloProcess()

- Se suscribe como receptor de eventos tipo HELLO_RECEIVED y crea una lista de mensajes.

void eventProcessing (int eventType, byte[] eventInformation)

- Método encargado de procesar el paquete recibido. Se irán procesando los fragmentos y si durante este proceso se completa un mensaje se procede a su reensablado. Con el mensaje JSON al completo, se llamará al método *addDevice()* que procesará y añadirá el dispositivo al conjunto de dispositivos finales. Por último, se eliminará el mensaje de la lista de mensajes.

String reassemble (int pos)

- Devuelve el mensaje JSON completo.

int fragmentProcess(byte[] data)

- Se ocupa de procesar los fragmentos del mensaje JSON recibidos. Se irá obteniendo del array de bytes cada uno de los campos del paquete y a partir de la dirección origen (XBeeAddress) se comprobará si :
  - El fragmento pertenece a un mensaje que esta siendo reensamblado, se añadirá el fragmento a éste.
  - El fragmento no existe, se creará un nuevo mensaje, añadiendo la dirección origen y el formato del mensaje que se han obtenido del paquete recibido.
- En ambos casos se procesará el fragmento antes de ser añadido. Se quitará la cabecera y se creará un nuevo fragmento con solo la carga útil. Por último se añadirá este fragmento al mensaje.
- Devulve la posición del mensaje completado.
- Se prestará atención al bitFinal, ya que nos indica si se trata del último fragmento del mensaje.

int getIndexByAddress (byte[ ] xbeeAddress)

- Busca en la lista de mensajes si existe un mensaje con la dirección origen (XBeeAddress) pasada como parametro.
- Devuelve la posición del mensaje cuya dirección origen (XBeeAddress) coincide con la pasada como parámetro o valor de la primera posición libre.

Message. Clase que almacena cada uno de los mensajes JSON publicados por los dispositivos finales. Formado por una lista de fragmentos, el número total de fragmentos y el formato del mensaje (JSON…).

Message()

- Crea una lista de fragmentos que formarán el mensaje completo.

boolean verifyMessageComplete()

- Comprueba si un mensaje contiene todos los fragmentos y por lo tanto está completo.

String getMessagePayload()

- Devuelve el mensaje JSON completamente reensablado.

isXBeeAddressEquals(byte[] xbeeAddress1)

- Compara dos direcciones de tipo XBeeAddress.

Métodos getter y setter

Fragment. Clase que almacena los fragmentos en los que se dividió el mensaje JSON por parte de los dispositivos finales.

Fragment(byte[] data, int length)

Métodos getter y setter

ResponseProcess. Clase que hereda de la clase *EventManager* y que actúa como suscriptor del evento RESPONSE_RECEIVED. Es la encargada de procesar los paquetes que contienen la respuesta por parte de los dispositivos finales en contestación a una petición realizada previamente.

ResponseProcess()

- Se suscribe como receptor de eventos tipo RESPONSE_RECEIVED.

void eventProcessing (int eventType, byte[] eventInformation)

- Método encargado de procesar el paquete recibido. Se quitará la cabecera para obtener la carga útil del mensaje que contiene la informacion del sensor o actuador. Finalmente ésta se imprimirá en la ventana de la interfaz gráfica correspondiente.

DeviceManager. Clase que actúa como gestor de dispositivos finales. Se trata de un conjunto en el que se pueden añadir, eliminar, buscar, modificar dispositivos y comprobar si existe un dispositivo con un cierto "friendly name". Implementa la interfaz *IDeviceManager*.

DeviceManager()

- Crea el conjunto de dispositivos finales.

HashSet<Device> devicesList ()

- Devuelve el conjunto de dispositivos finales.

boolean addDevice (String json, String mimeType)

- Permite añadir un dispositivo final en el conjunto de dispositivos. En función del mimeType recibido se procesará el mensaje de una u otra manera. En esta aplicación se ha trabajado con mensajes de tipo JSON.
- Se parseará el mensaje JSON y se obtendrá el dispositivo.
- Se comprobará que el dispositivo no existe ya en el conjunto. Si es así se añadirá éste al conjunto y por último se actualizará el árbol de la interfaz gráfica donde se representan los dispositivos finales.

void deleteDevice (String deviceID)

- Permite eliminar un dispositivo del conjunto por deviceID.

Device searchDevice (String key)

- Facilita la búsqueda de un dispositivo en el conjunto bien sea por deviceID o friendlyName. Si se ha encuentra el dispositivo en el conjunto se devuelve y si no null.

void modifyDevice (Device device, String newFriendlyName)

- Modifica el "friendly name" del dispositivo pasado como argumento por el nuevo "friendly name" facilitado.

boolean existsFriendlyName (String newFriendlyName)

- Comprueba que el "friendly name" introducido por el usuario no exista ya en el conjunto de dispositivos finales.

Device. Clase que define a un dispositivo final. Está formado por el tipo de dispositivo, el identificador único de cada dispositivo, parámetros, servicios y "friendly name".

Device (String deviceType, String deviceID, Parameters, List <Service> service)

- Crea un dispositivo con las caracrteríticas y servicios publicados por el dispositivo final.

boolean equals (Object o)

- Permite saber si un dispositivo es igual que otro por "friendly name".

void addService (Service[] service)

- Sirve para añadir una lista de servicios.

Métodos getter y setter

Arduino. Clase que representa un tipo de dispositivo final. Hereda de la clase Device.

Arduino (String deviceType, String deviceID, Parameters, List <Service> service, String xBeeAddress)

- Crea un dispositivo final de tipo Arduino. Se direfencia en la dirección de 64 bit, XBeeAddress.

Métodos getter y setter

Parameters. Clase que almacena la lista de parámetros que contiene el mensaje JSON describiendo características del dispositivo final. Está formado por una lista de atributos.

void addParameters(Attribute[] attribute)

- Permite añadir una lista de atributos a la lista de parametros.

Métodos getter y setter

Service. Clase que almacena los servicios que contiene el mensaje JSON los cuales son ofrecidos por el dispositivo final. Contiene un nombre, una lista de atributos y el valor devuelto.

void addParameters(Attribute attribute)

- Permite añadir atributos a la lista de atributos.

Métodos getter y setter

Attribute. Clase que representa un atributo contenido en el mensaje JSON tanto en los parámetros como en los servicios.

Métodos getter y setter

JSON. Clase que se encarga de parsear los mensajes con formato JSON. Contiene las clases: *documentJSON*, *ParametersJSON*, *Services*, *ParametersServ*.

WMain. Clase que representa la ventana principal de la interfaz gráfica.

WTemperatureSensor. Clase que representa la ventana para el servicio de obtención de temperatura de un dispositivo final.

WLed. Clase que representa la ventana para el servicio de encendido y apagado de un LED en un dispositivo final.

WProgressBar. Clase que representa la ventana de espera que se muestra mientras se está obteniendo el resultado de la medida de temperatura del dispositivo final.

3.2.5. Manual de Usuario de la Aplicación

A continuación se presenta el manual de usuario de la aplicación desarrollada en este proyecto, en el que se pretende guiar paso a paso al usuario en el manejo de ésta.

La aplicación permite, una vez establecida la conexión entre los dispositivos finales y el coordinador, visualizar las características y servicios ofrecidos por los dispositivos finales disponibles en la red y realizar una serie de operaciones (peticiones, modificar el friendly name, borrar un dispositivo) que se explicarán a continuación. En el caso de uso se desarrollan dos servicios: obtener la temperatura de un determinado entorno y encender/apagar un LED.

3.2.5.1. Configuraciones y Montaje

Antes de poner en funcionamiento la aplicación se deberán realizar una serie de configuraciones, y a continuación proceder con el montaje de todas las partes del sistema.

- **Configuraciones.**
  - ✓ Los módulos XBee han de configurarse como se explica en el apartado 3.2.3.1, para que el coordinador pueda establecer la red y los dispositivos finales unirse a ésta, poniendo así en funcionamiento una comunicación inalámbrica.
  - ✓ Deben cargarse los sketches en los correspondientes Arduinos.
  - ✓ En el PC debe estar el entorno de trabajo (Eclipse, NetBeans…) abierto con el proyecto de la aplicación.
- **Montajes.**
  - ✓ Dispositivo final con sensor de temperatura.

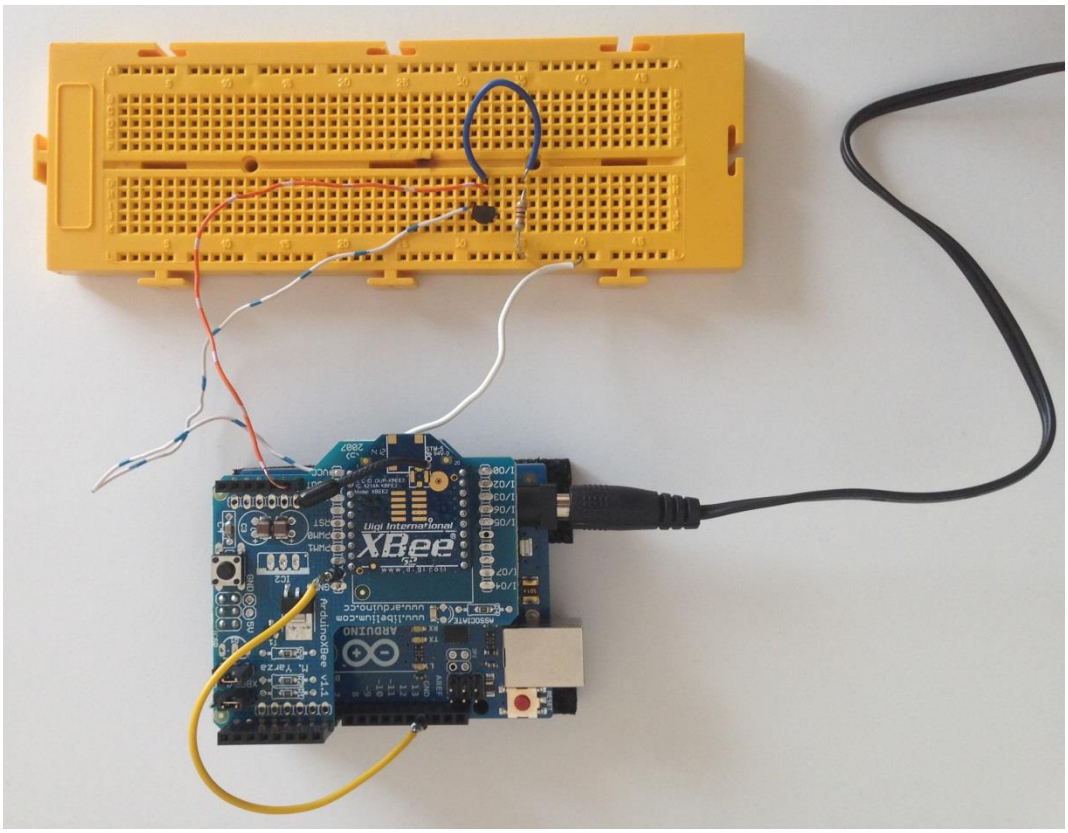


Figura 75. Dispositivo Final con Sensor de Temperatura conectado a un Alimentador electrónico.

✓ Dispositivo final con actuador LED.

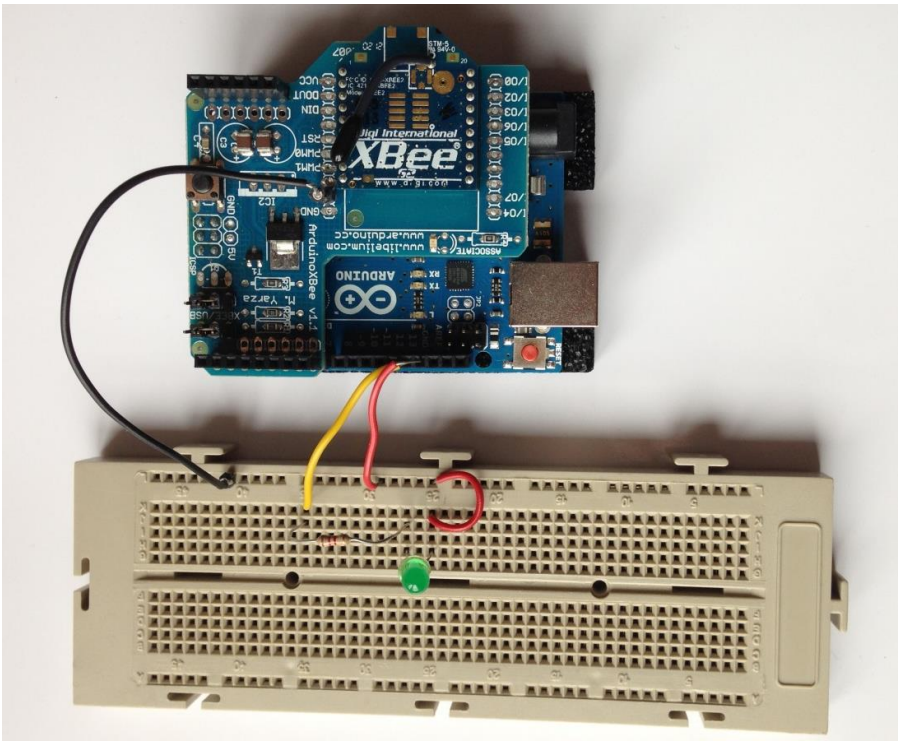


Figura 76. Dispositivo Final con LED conectado a un Alimentador electrónico.

✓ Coordinador.

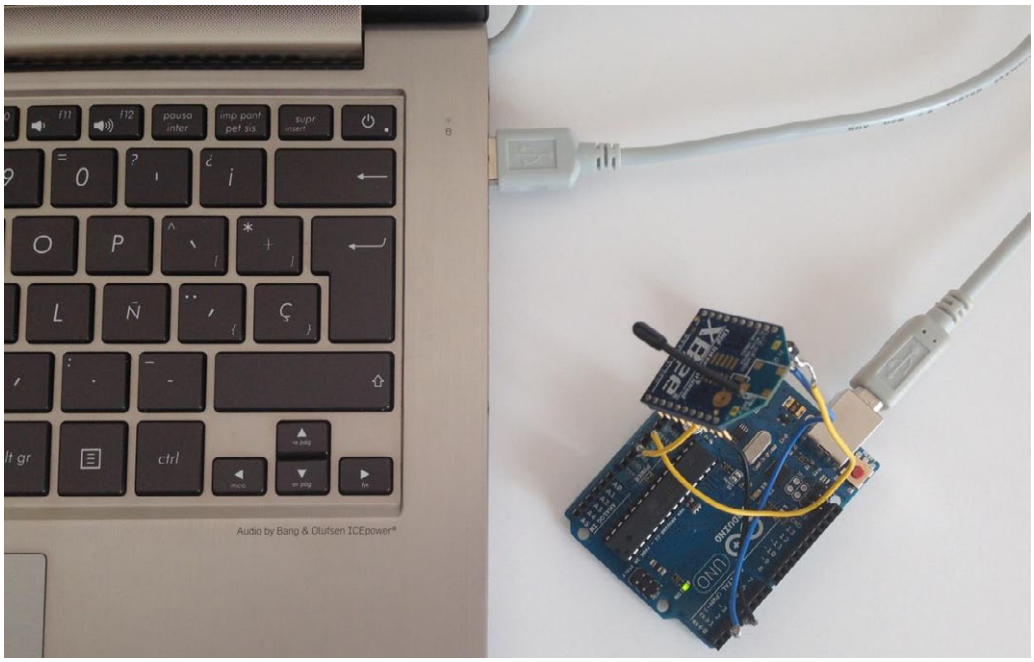


Figura 77. Coordinador conectado vía USB al PC.

- ✓ Sistema completo.

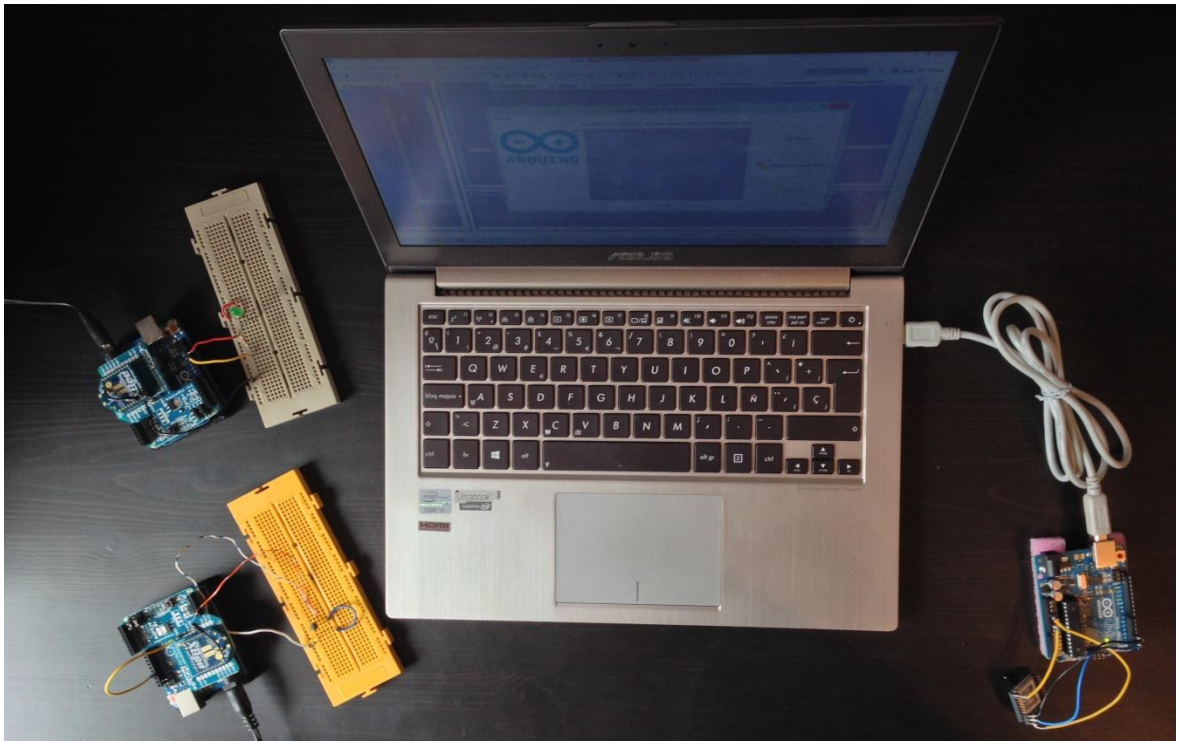


Figura 78. Sistema completo de la aplicación.

3.2.5.2. Funcionamiento de la Aplicación.

Una vez realizadas las configuraciones y los montajes necesarios, ya se puede proceder a poner en funcionamiento la aplicación. En el PC se ejecutará la aplicación y mostrará la pantalla principal, como se puede ver en la Figura 79.

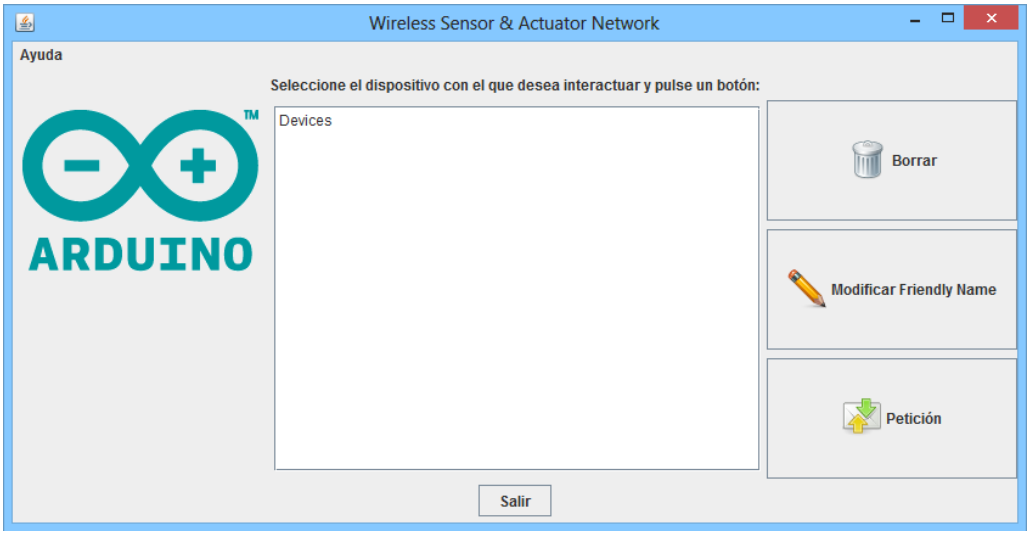


Figura 79. Pantalla principal de la aplicación.

La pantalla está formada por tres botones "Borrar", "Modificar Friendly Name" y Petición", que en los siguientes apartados se explicará su funcionamiento, y en la parte central aparece un árbol de dispositivos vacío, a medida que los dispositivos se unan a la red irán mostrándose en éste, como se aprecia en la Figura 80.

Figura 80. Descubrimiento de dispositivos finales.

Los dispositivos finales envían un JSON que los identifica, facilitando características propias de cada uno de ellos y los servicios que proporcionan. En la Figura 81 se muestra el árbol completo de un dispositivo final.

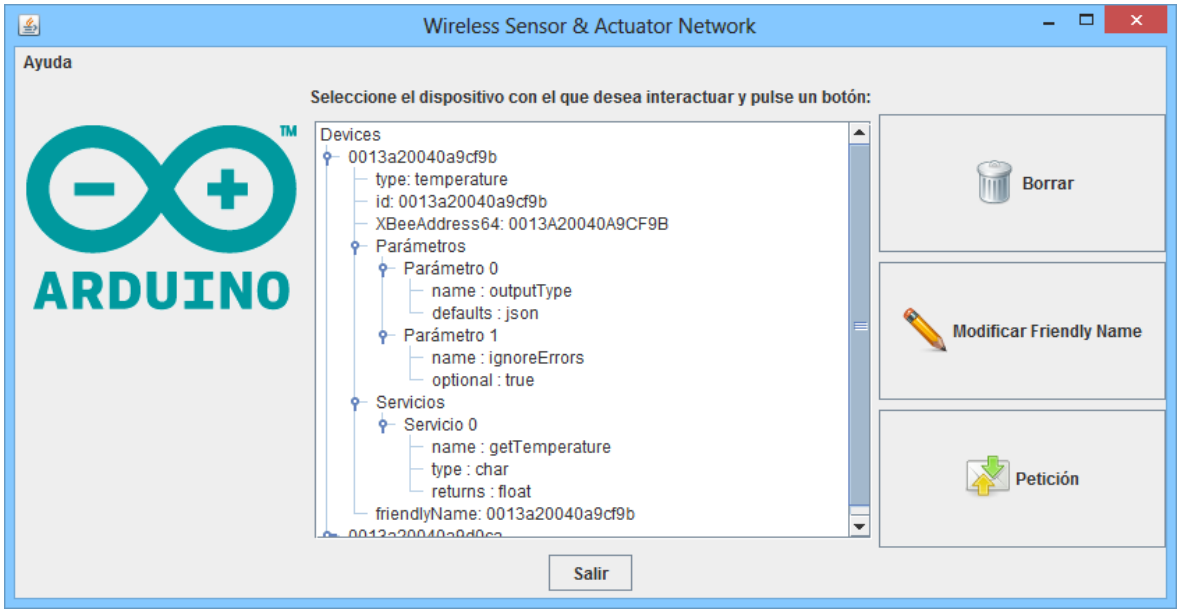


Figura 81. Características y servicios de un dispositivo final.

3.2.5.3. Funcionalidades

La aplicación ofrece tres funcionalidades: realizar una petición a un dispositivo final, modificar el friendly name y borrar un dispositivo final. Éstas podrán usarse sobre un dispositivo una vez que éste esté visible en el árbol central, ya que significará que el dispositivo se ha unido a la red y ha informado de sus características y servicios.

A continuación se da una explicación detallada y gráfica de cómo hacer uso de estas funcionalidades.

- **Asignar Friendly Name a un Dispositivo Final.**

Buscando la manera de hacer la aplicación más amigable y cercana al usuario, se ofrece la opción de modificar el nombre inicial con el que aparece el dispositivo en la pantalla principal por uno más reconocible por el usuario.

Para realizar cualquiera de las tres funcionalidades ofrecidas se seguirán los mismos pasos:

- ✓ Seleccionar el dispositivo con el que se desea interactuar del árbol central de dispositivos finales.
- ✓ Pulsar el botón correspondiente. En este caso "Modificar Friendly Name", situado en la parte derecha de la interfaz.

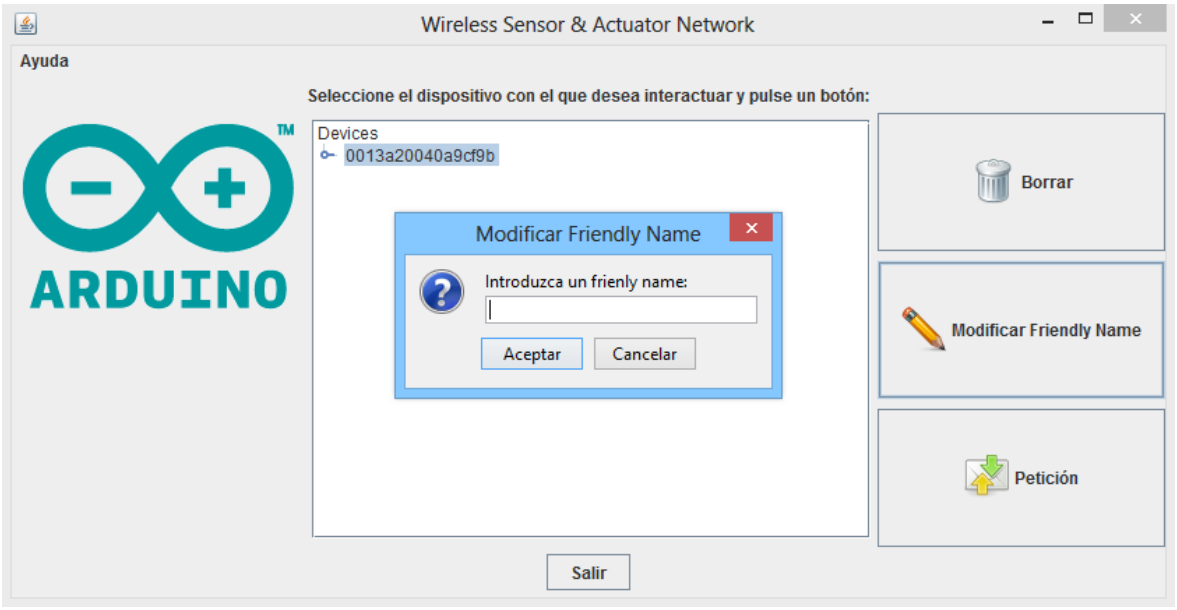


Figura 82. Modificar Friendly Name.

Aparecerá en pantalla una nueva ventana indicándole que introduzca el nuevo "friendly name", como se aprecia a modo de ejemplo en la Figura 83.

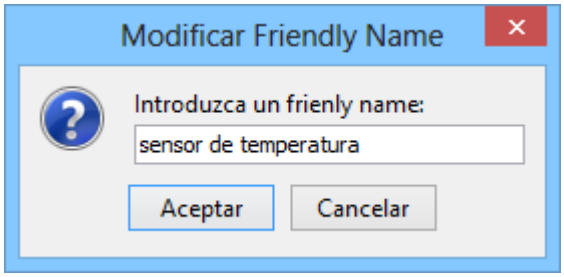


Figura 83. Ejemplo friendly name.

Una vez tenga escrito el "friendly name" y pulse el botón "Aceptar", aparecerá en pantalla una ventana de confirmación, como se ilustra en la Figura 84.

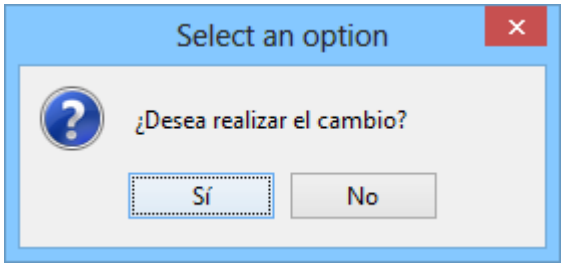


Figura 84 .Confirmación de cambio.

En la Figura 85 se puede ver el resultado final de haber modificado los "friendly name" de todos los dispositivos finales actualmente disponibles.

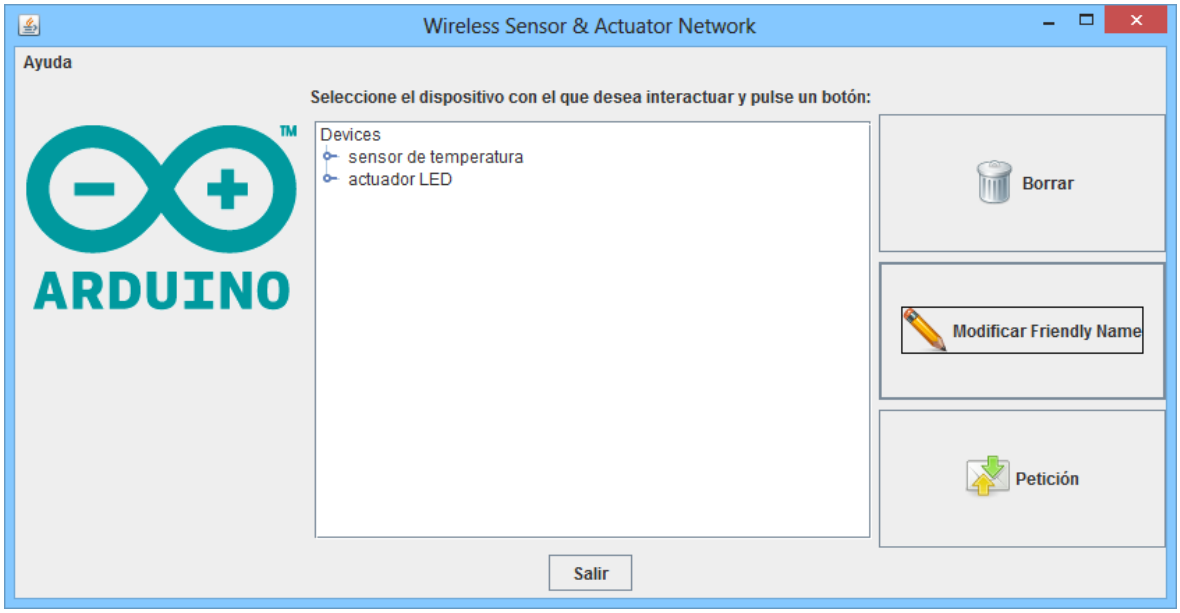


Figura 85. Pantalla principal con friendly name modificado.

- **Envío de Petición a un Dispositivo Final.**

Hasta el momento la aplicación está dotada de dos dispositivos finales, los cuales proveen dos servicios diferentes.

Del mismo modo que se hizo para la modificación del "friendly name" se deberá:

- ✓ Seleccionar el dispositivo con el que se desea interactuar del árbol central de dispositivos finales.
- ✓ Pulsar el botón correspondiente. En este caso "Petición", situado en la parte derecha de la interfaz.

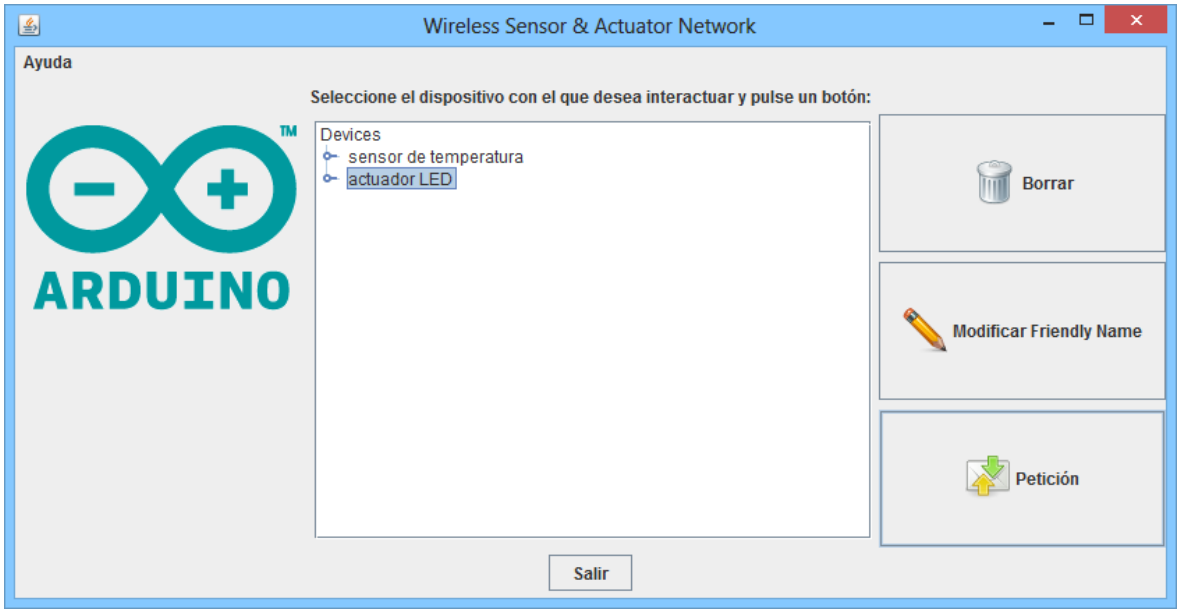


Figura 86. Pantalla principal, realizando petición.

En primer lugar se detallará el funcionamiento del control remoto de un actuador, en este caso de un LED. Tras haber pulsado el botón de "Petición" aparecerá en pantalla una ventana que permitirá encender o apagar un LED situado en el dispositivo final utilizando los botones "ON" y "OFF" respectivamente.

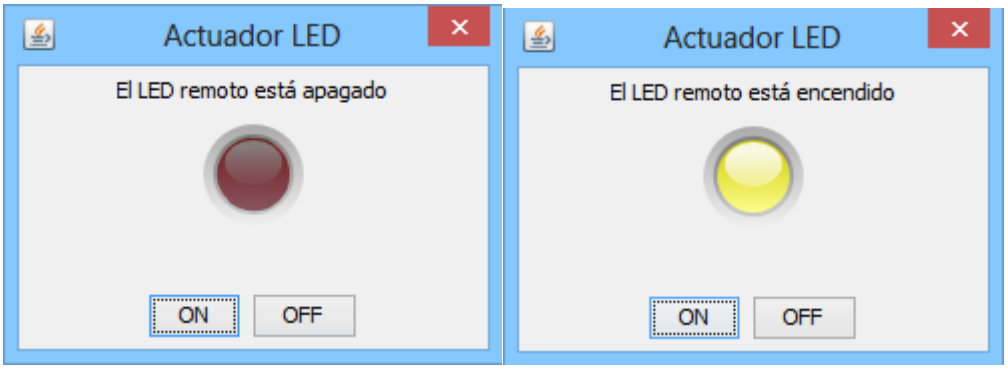


Figura 87. Ventana Actuador LED, "ON" y "OFF".

En segundo lugar, se explicará la manera de obtener la temperatura del entorno donde se encuentra situado el dispositivo final. Una vez haya sido pulsado el botón "Petición" se presentará al usuario una ventana con la posibilidad de obtener el resultado en Celsius o Fahrenheit tal y como se muestra en la Figura 88.

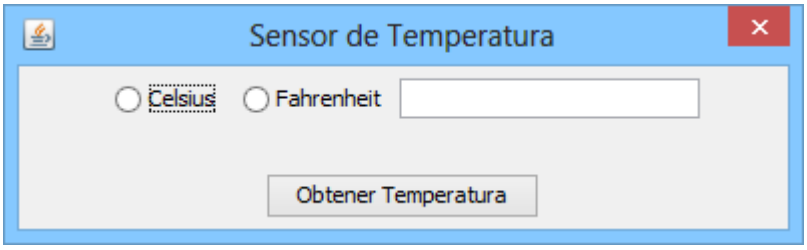


Figura 88. Ventana Sensor de Temperatura.

Para hacer la petición se debe seleccionar una de las dos opciones, Celsius o Fahrenheit, y a continuación pulsar el botón "Obtener Temperatura". El resultado se mostrará en la propia ventana tal y como se muestra en la Figura 89.

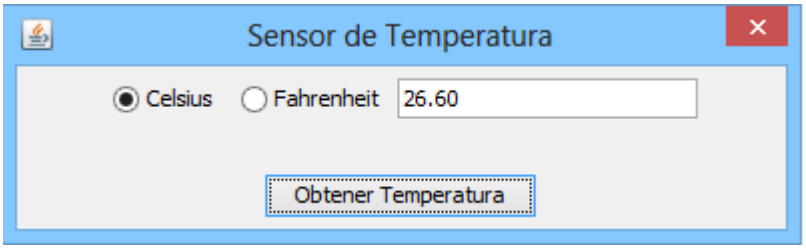


Figura 89. Ventana Sensor de Temperatura, resultado final.

Durante el periodo de tiempo que se espera la respuesta a la petición realizada, se muestra una ventana de espera, como se ve en la siguiente Figura 90.

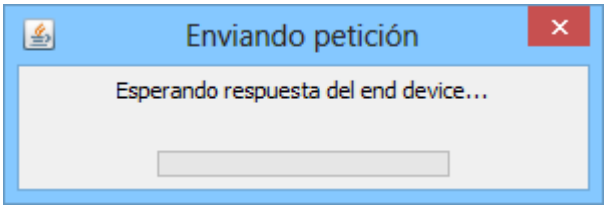


Figura 90. Ventana de Espera.

- **Borrar Dispositivo Final.**

La aplicación da la posibilidad de borrar cualquier dispositivo final del árbol de "Devices", además éste es eliminado del conjunto de dispositivos.

Como se ha hecho en los dos casos anteriores, se deberá:

- ✓ Seleccionar el dispositivo con el que se desea interactuar del árbol central de dispositivos finales.

- ✓ Pulsar el botón correspondiente. En este caso "Borrar", situado en la parte derecha de la interfaz.

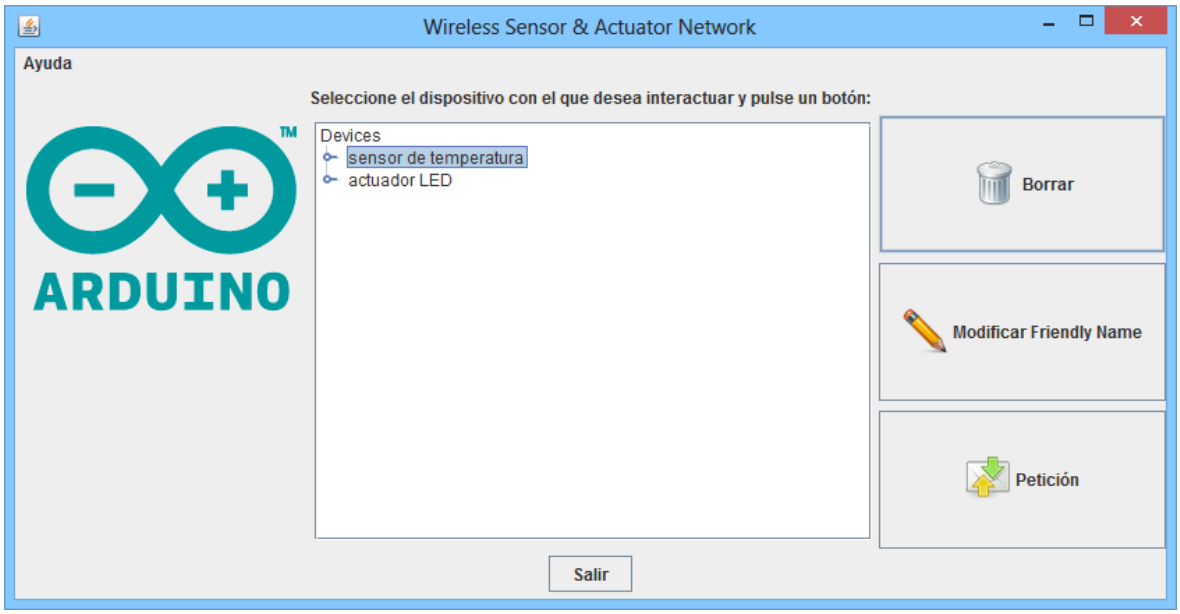


Figura 91. Pantalla principal, borrando dispositivo.

En la Figura 92 se muestra un ejemplo, en el que se ha borrado el dispositivo "sensor de temperatura", quedando únicamente el dispositivo "actuador LED" en el árbol central.

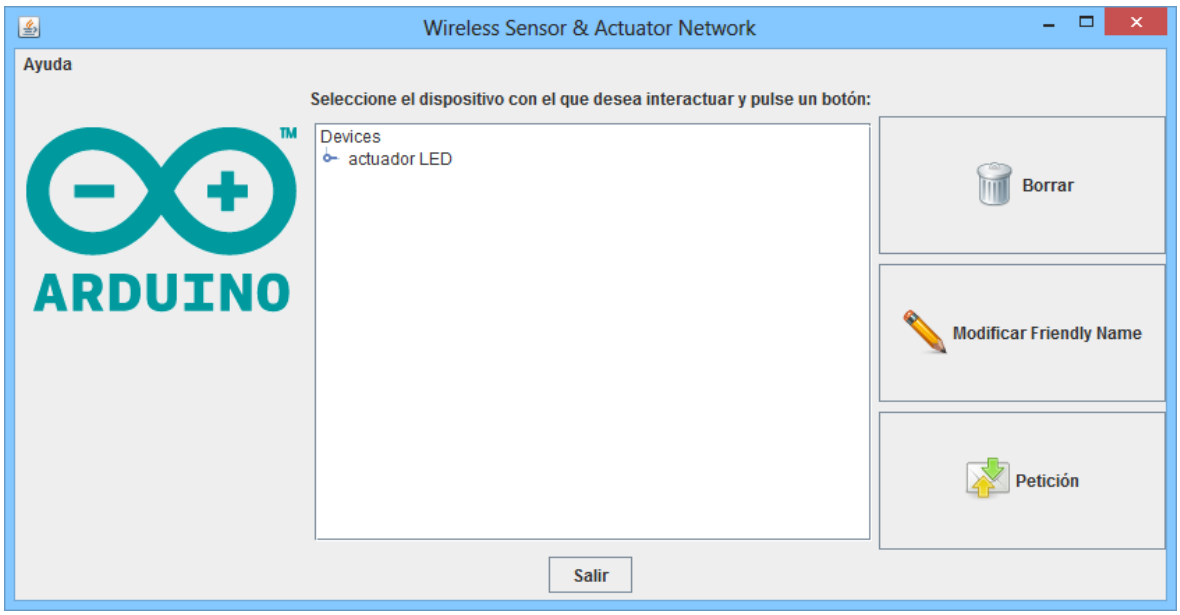


Figura 92. Pantalla Principal, dispositivo borrado.

Capítulo 4:

Conclusión

4.Conclusión

Teniendo en cuenta los objetivos planteados en el apartado 1 se analizará en este capítulo si se han conseguido llevar a cabo, y extraer ciertas conclusiones que permitan valorar el trabajo realizado en este proyecto.

El primero de ellos era integrar una nueva plataforma hardware en un sistema ya existente, que utiliza un middleware orientado a servicios llamado nSOM (nano Service-Oriented Middleware) desarrollado en la UPM y que hace uso de la mota SunSPOT.

En apartados anteriores se presentaron algunas de las múltiples plataformas hardware existentes en el mercado, de entre las cuales se seleccionó la que mejor se adaptaba a las necesidades de este proyecto. Finalmente se optó por trabajar con la plataforma Arduino UNO (rev3), la cual ha cumplido con las expectativas de este proyecto, presentado retos importantes que la hacían más atractiva para la investigación a desarrollar en este proyecto.

Una vez hecha la elección se procedió a la realización del diseño del sistema de este proyecto, teniendo en cuenta que se tenía que lograr en un futuro la integración de la plataforma en un sistema ya existente. Dicho sistema cuenta con una propuesta de arquitectura de middleware llamada nSOM. Está arquitectura está diseñada para ser usada en redes WSAN. Utiliza agentes y está inspirada en “Service-Oriented Architecture (SOA)” y en el paradigma “Service-Oriented Computing (SOC)”. El principal objetivo de nSOM es proporcionar servicios de gestión a los usuarios finales, para ello formará parte de la capa “Service-Oriented Software Platform” del modelo de arquitectura WSAN-SOA. Siguiendo el paradigma SOA en el que se inspira nSOM se ha desarrollado el sistema de este proyecto, incluyendo servicios de publicación de dispositivos finales, envío de peticiones y recepción de respuestas.

Una de las metas a conseguir era el descubrimiento de dispositivos finales por medio de la publicación de mensajes JSON. Para poder realizar el envió de este mensaje, fue necesario realizar fragmentación en los dispositivos finales, tal y como se vio en el apartado 3.1.6.3. Inicialmente, la lógica de reensamblado se localizaba en el nodo coordinador, pero debido al problema de la falta de memoria expuesto en el apartado 3.1.6.4, no se pudo llevar a cabo.

Para suplir esta carencia, y como ya se explicó en el apartado 3.1.6.3, se trasladó dicha lógica a un dispositivo con mayor capacidad, el PC. Así que se puede concluir que, a pesar de los problemas encontrados y de no ser éste el planteamiento inicial, se ha conseguido la publicación de las características y servicios de cada dispositivo final.

Normalmente el comportamiento que muestran las aplicaciones WSAN es el siguiente: los dispositivos finales realizan mediciones sobre el medio y envían dichas muestras constantemente fuera de la red de sensores y actuadores a través de un gateway a una estación base, donde la información será almacenada y tratada. El sistema definido y desarrollado en este proyecto difiere de este modelo, buscando la orientación a servicios. Se ha pretendido que sea el usuario quien pueda solicitar la información que desea o actuar sobre un determinado dispositivo cuando desee. Dicho objetivo ha sido conseguido, ya que el usuario es capaz, a través de la interfaz gráfica, de visualizar los dispositivos finales disponibles y realizar peticiones a cualquiera de ellos obteniendo la información requerida en ese instante.

Cabe destacar el desarrollo de un protocolo E2E propio que define las comunicaciones entre los dispositivos finales y el PC, basado en el equivalente de nSOM. Se ha observado la necesidad de desarrollar este protocolo para poder dar soporte a la funcionalidad de intercambio de mensajes.

Un requisito importante a la hora de diseñar y desarrollar este sistema era plantear una solución software genérica y reutilizable, permitiendo la incorporación de cualquier plataforma hardware al sistema siendo ésta transparente para el desarrollador. Se puede afirmar que se ha alcanzado dicho requisito.

Las plataformas hardware diseñadas para redes WSAN son dispositivos de baja capacidad, lo que supone ser muy cuidadosos con la programación ya que de lo contrario, esto podría suponer que el nodo dejase de estar operativo por falta de memoria. Durante el desarrollo de este proyecto se ha tenido muy en cuenta cumplir estas premisas, ya que el nodo con el que se ha trabajado contaba con menos memoria de la que normalmente están dotadas estas plataformas. Un ejemplo sería tener que evitar la utilización de ciertas clases como String puesto que se ha observado que sobrescribe zonas de memoria incorrectas siendo su comportamiento impredecible.

A lo largo de la implementación de la aplicación se han tenido que realizar numerosas pruebas y correcciones sobre el código. Uno de los grandes inconvenientes fue la transformación de los datos a otro tipo de variables. Por ejemplo, en la secuencia de envío de la temperatura se han realizado:

- Los datos tomados por el sensor son almacenados en una variable de tipo float (no codificado en ASCII) que han de ser convertidos a una variable de tipo uint8_t con codificación ASCII. Esto es debido a que se busca una mayor compatibilidad con el protocolo nSOM.

- Una vez estos datos hayan llegado al PC será necesaria una conversión a un array de bytes y posteriormente a un objeto String.

Por lo que se puede concluir que el tratamiento de tipos de datos es un aspecto delicado y al que hay que tomar en especial consideración a la hora de desarrollar.

En cuanto a las pruebas realizadas se decidió dividirlas en unitarias e integradas, previendo que en este último caso, encontrar un error de ejecución en el sistema sería menos eficiente y más costoso:

- Pruebas unitarias: éstas se realizaron en los dispositivos finales, el coordinador y el PC, comprobando que cada uno de ellos por separado funcionaban correctamente.
  - ✓ En los dispositivos finales se verificó que la función de fragmentación se realizase sin problemas, así como la recepción de peticiones y el envío de respuestas con la información solicitada por el usuario.
  - ✓ En el coordinador se trabajó en las comunicaciones serie con el PC y en las inalámbricas con los dispositivos finales.
  - ✓ En el PC se han hecho pruebas para un apropiado desarrollo de la interfaz gráfica.
- Pruebas de integración: una vez aprobadas las pruebas unitarias se validó que el conjunto de elementos del sistema funcionasen de manera combinada como un todo.

Para finalizar se han analizado los tiempos de transmisión y recepción de paquetes. Se presentan dos casos de estudio:

Tiempo que tarda en realizarse una transmisión de una petición y recepción de la respuesta correspondiente.

- Experimento: En este caso de estudio se va a medir el tiempo que trascurre desde que se envía una petición hasta que se recibe la respuesta correspondiente. Para ello se utilizará el servicio “Obtener temperatura”, en el cual un usuario selecciona un parámetro (Celsius o Fahrenheit) y solicita la temperatura, tras lo cual se quedará esperando a que llegue el resultado solicitado que le será mostrado en la ventana.

  Las mediciones se llevarán a cabo haciendo uso del método *nanoTime()* de la clase System. Este devuelve el valor actual, en nanosegundos, de la fuente de

tiempo de alta resolución de la Java Virtual Machine que está ejecutándose. Los valores devueltos por este método adquieren sentido sólo cuando se calcula la diferencia entre dos de estos valores, obtenidos dentro de la misma instancia de una máquina virtual Java.

- ✓ Se definirá una variable de tipo *long* que almacene el tiempo de referencia: *long start.*
- ✓ El primer tiempo se tomará en la función *writeData(byte[] data)* de la clase SerialCommunications, encargada de escribir los datos en el puerto serie.

  long start = System.nanoTime();

  El segundo valor se tomará en la función *serialEvent (SerialPortEvent oEvent)* de la clase SerialCommunications, encargada de manejar los eventos detectados en un puerto serie, en este caso si hay datos disponibles en el puerto serie. Se imprimirá por pantalla el valor de la diferencia entre ambos valores, tras haber recibido el paquete respuesta completo.

  System.out.println (System.nanoTime()-start));
- ✓ Este proceso se repetirá 50 veces variando la distancia entre el nodo coordinador y el dispositivo final.

- ➢ Análisis de resultados: se van a presentar los diferentes casos dependiendo de la distancia tomada. En primer lugar se mostrará una tabla con las muestras tomadas y seguidamente las gráficas que relacionan estos valores mediante la media, desviación estándar y la distribución normal. Se analizará cada uno de los casos y, por último, se extraerán ciertas conclusiones.

Caso 1. Distancia de un metro sin obstáculos. En este caso se colocará el dispositivo final a un metro del coordinador sin obstáculos de por medio, y se procederá a realizar las medidas.

Muestras					
1	0,67814883	18	0,63456436	35	0,50450594
2	0,49178168	19	0,33634292	36	0,43059497
3	0,40956772	20	0,24696812	37	0,50063742
4	0,61465874	21	0,36503757	38	0,22611388
5	0,62222498	22	0,32345957	39	0,41667064
6	0,2237108	23	0,40880897	40	0,52676259
7	0,52126954	24	0,49370165	41	0,51686733
8	0,20755201	25	0,51774044	42	0,3753397
9	0,60994277	26	0,49220908	43	0,55742779
10	0,20100991	27	0,55442138	44	0,54052711
11	0,5433671	28	0,41915288	45	0,46687785
12	0,23131589	29	0,38515872	46	0,42509532
13	0,2710223	30	0,3101657	47	0,4302988
14	0,25966892	31	0,55659133	48	0,38054904
15	0,22616153	32	0,4189916	49	0,42677263
16	0,34609523	33	0,51927553	50	0,53757128
17	0,25610243	34	0,54693726		

Tabla 3. Muestras Caso 1 (Distancia un metro sin obstáculos) en segundos.

A partir de las muestras tomadas se calculó el promedio y la desviación estándar:

Desviación	Media
0,12849728	0,43011479

Tabla 4. Desviación estándar y Media del Caso 1 (segundos).

En el gráfico de la Figura 93 se distinguen el conjunto de muestras tomadas, la media de todas ellas y la desviación estándar que sufren. Observando el gráfico vemos como todas las muestras se concentran en torno al medio segundo con apenas una desviación estándar de 0,12849728. Por lo que se puede concluir que el tiempo transcurrido desde que se envía una petición hasta que se recibe la respuesta es muy constante proporcionando un servicio eficaz al usuario.

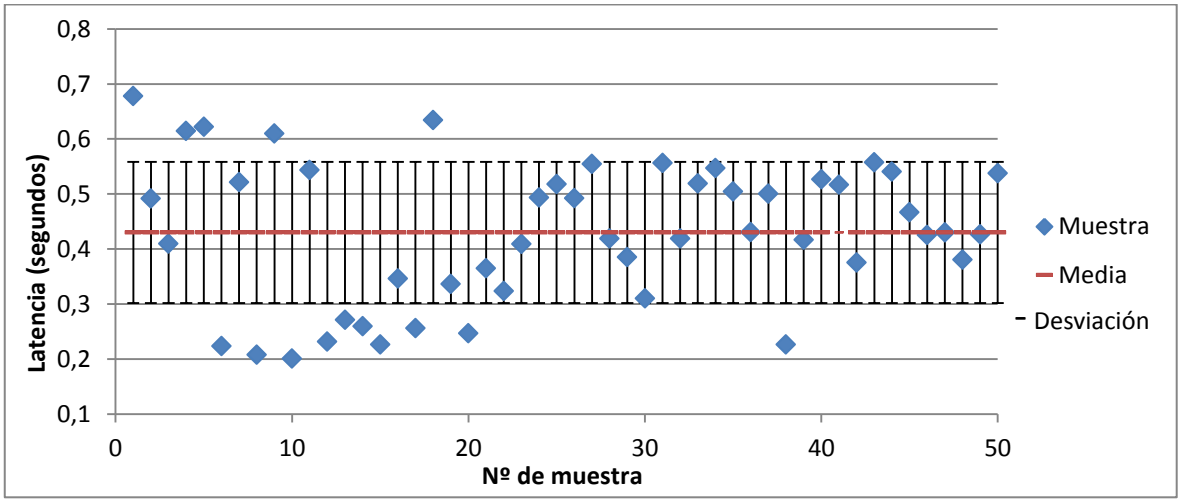


Figura 93. Gráfico de Línea con marcadores y media del Caso 1 (segundos).

Por último se presenta el gráfico de la Figura 94 en cual obtenemos una distribución normal a partir de los resultados obtenidos en la Tabla 3 y Tabla 4, es decir, la mayoría de los valores están cercanos al promedio mientras que una menor cantidad de valores están ubicados en los extremos.

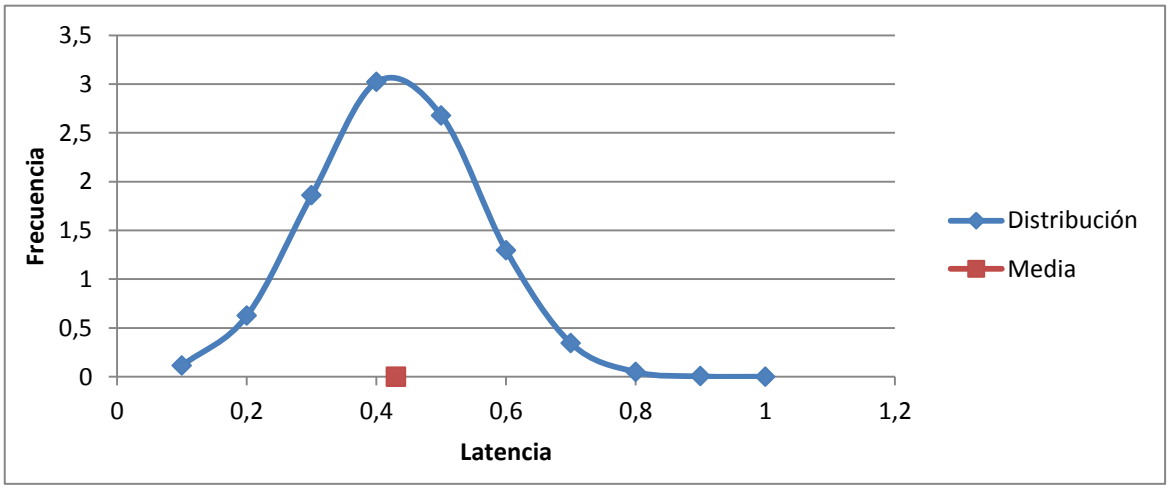


Figura 94. Gráfico de Dispersión y media del Caso 1 (segundos).

Caso 2. Distancia de 9 metros sin obstáculos. Se alejará el dispositivo final a 9 metros del coordinador sin que haya obstáculos de por medio, y se procederá a realizar las muestras.

Muestras					
1	0,67297321	18	0,35845294	35	0,51036481
2	0,63735084	19	0,34033826	36	0,48498813
3	0,47983376	20	0,35527793	37	0,43480365
4	0,3494997	21	0,56839556	38	0,18392009
5	0,64584664	22	0,53607871	39	0,44661081
6	0,44273129	23	0,63834272	40	0,53220506
7	0,50960826	24	0,21416669	41	0,43457272
8	0,46762853	25	0,43603231	42	0,50488129
9	0,60153861	26	0,59300836	43	0,59089853
10	0,58081193	27	0,56670066	44	0,51504706
11	0,5466169	28	0,63266419	45	0,68874053
12	0,60880281	29	0,36055985	46	0,56870933
13	0,23887626	30	0,44384046	47	0,57529982
14	0,39149482	31	0,43410354	48	0,57374053
15	0,39822167	32	0,2169781	49	0,40380929
16	0,36544663	33	0,57314013	50	0,41760386
17	0,27185803	34	0,48706278		

Tabla 5. Muestras Caso 2 (Distancia 9 metros sin obstáculos) en segundos.

A partir de las muestras realizadas se calculó el promedio y la desviación típica:

Desviación	Media
0,12542216	0,47660957

Tabla 6. Desviación estándar y Media del Caso 2 (segundos).

En los siguientes gráficos se puede observar que aun habiendo aumentado la distancia a 9 metros no se detectan cambios significativos respecto al Caso 1. Tanto la media como la desviación estándar son muy similares a las obtenidas en el Caso 1, obteniendo también una distribución normal. Se puede concluir que sigue siendo un servicio eficiente.

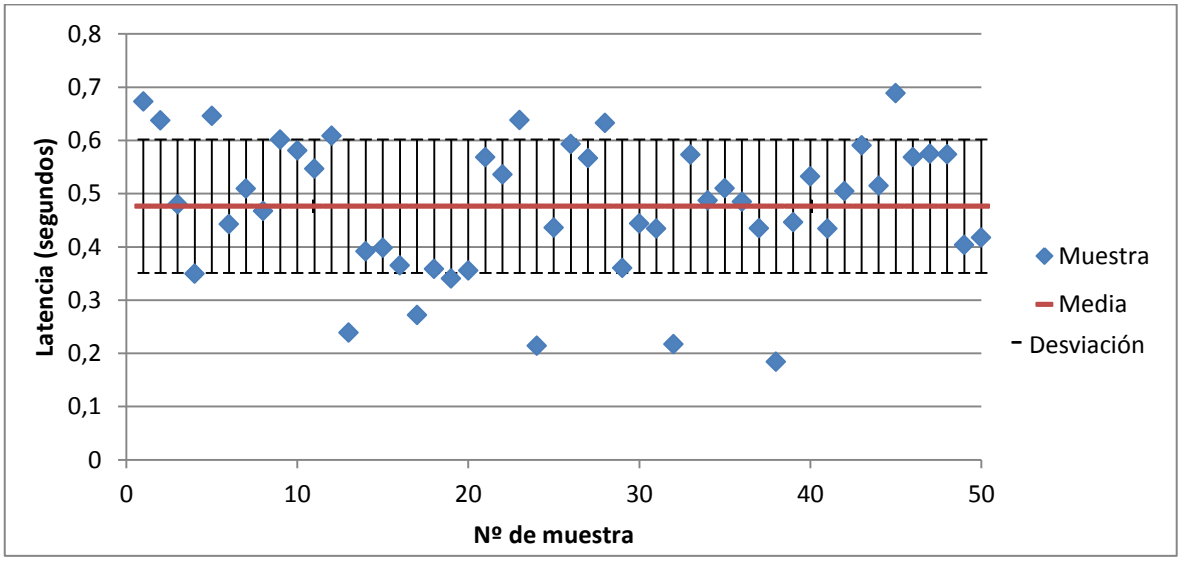


Figura 95. Gráfico de Línea con marcadores y media del Caso 2 (segundos).

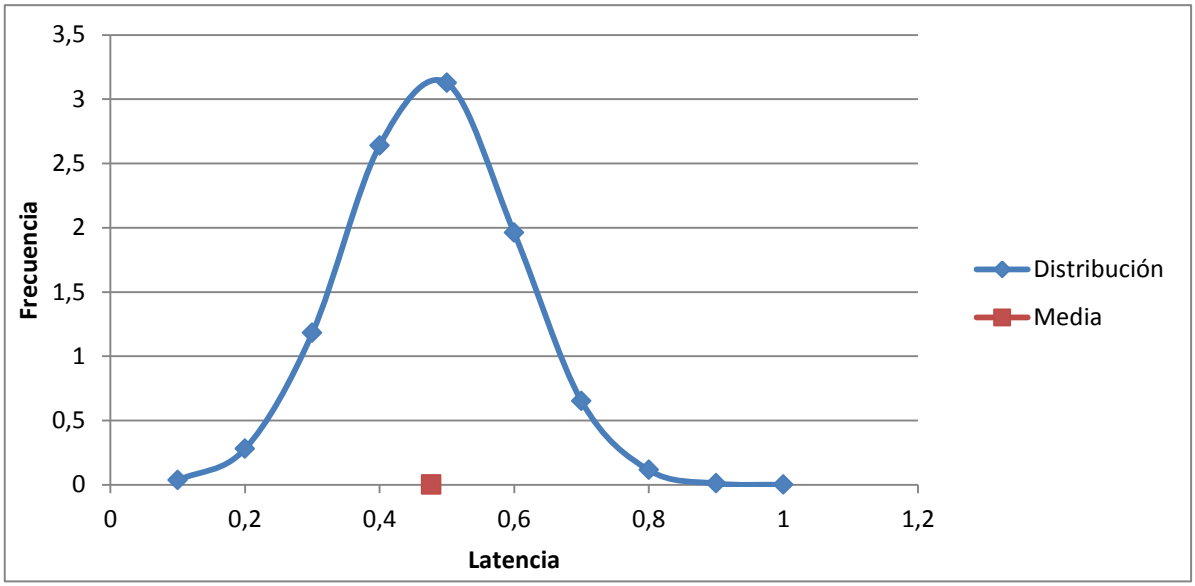


Figura 96. Gráfico de Dispersión y media del Caso 2 (segundos).

Caso 3. Distancia de 50 metros sin obstáculos. Por último, se posicionará el dispositivo final a 50 metros del coordinador sin obstáculos de por medio, y se procederá a realizar las muestras.

Muestras					
1	0,41612229	18	0,45866796	35	0,45202102
2	0,595333	19	0,40007859	36	0,47416477
3	0,19950194	20	0,41769183	37	3,28822795
4	0,31025441	21	0,41421625	38	0,40992327
5	0,26407333	22	0,25554529	39	0,47005213
6	0,39303065	23	0,38000875	40	0,4334247
7	0,45123661	24	0,42793238	41	0,48058298
8	0,56391711	25	0,58780782	42	0,42159994
9	0,57240997	26	0,54928168	43	0,53100865
10	0,54319629	27	0,36744724	44	0,53853677
11	0,59725663	28	0,31352692	45	0,49517003
12	0,28633658	29	0,38592626	46	0,54612352
13	0,47573139	30	0,48641179	47	0,58064698
14	0,50732908	31	0,46695629	48	3,30939009
15	0,46856982	32	0,48908098	49	0,32756195
16	0,61738951	33	0,56727467	50	0,42616417
17	0,4893163	34	0,47494112		

Tabla 7. Muestras Caso 3 (Distancia 50 metros sin obstáculos) en segundos.

A partir de las muestras tomadas se calculó el promedio y la desviación estándar:

Desviación	Media
0,5712285	0,56756799

Tabla 8. Desviación estándar y Media del Caso 3 (segundos).

En el gráfico de la Figura 97 se observa que en este caso se sigue manteniendo la misma concentración de muestras en torno al medio segundo pero hay que destacar la diferencia en el valor de la desviación estándar, siendo superior al de los casos anteriores. Este incremento se debe a que en ciertas ocasiones el tiempo de envío y recepción de peticiones y respuestas es muy superior a la media. Todo esto está motivado por el notable aumento en la distancia entre coordinador y dispositivo final. Pese a estas excepciones, se puede considerar que sigue habiendo un buen rendimiento en las comunicaciones.

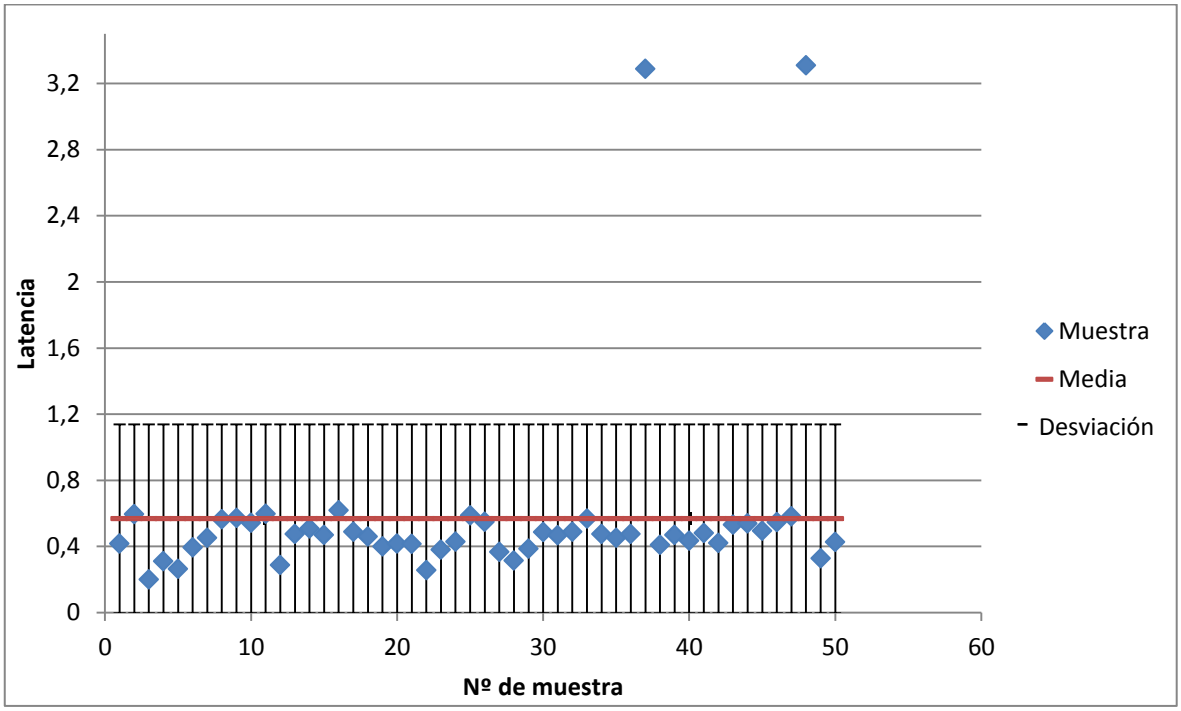


Figura 97. Gráfico de Línea con marcadores y media del Caso 3 (segundos).

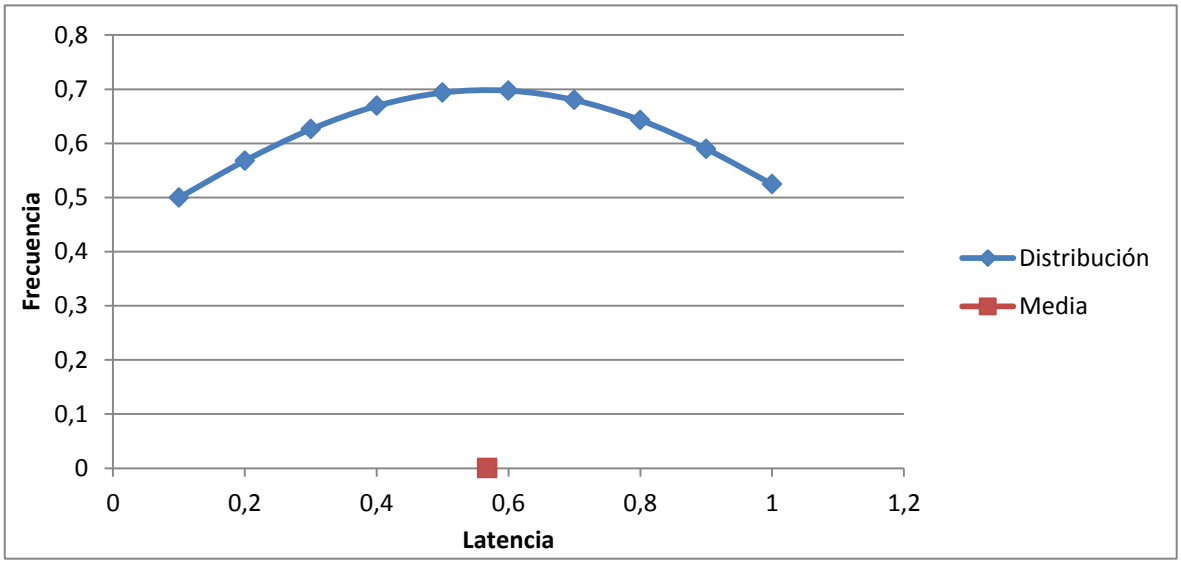


Figura 98. Gráfico de Dispersión y media del Caso 3 (segundos).

- Conclusiones: teniendo en cuenta el anterior análisis se concluye que el tiempo que tarda en llegar un paquete Request al dispositivo final enviado desde el PC más el tiempo de retorno del paquete Response, es de aproximadamente medio segundo. Sólo en ciertas ocasiones en las que las distancias eran muy larga entre éstos se observó una latencia superior a los 3 segundos y pérdidas de paquetes entorno al 4%. También se realizaron pruebas con obstáculos entre el

dispositivo final y el coordinador sin apreciarse cambios en los tiempos respecto a los mostrados en las pruebas sin obstáculos.

Tiempo que tarda en recibirse en el PC cada fragmento del paquete Hello, provenientes de los dispositivos finales, y su reensamblado.

- Experimento: En este caso de estudio se va a medir el tiempo que trascurre desde que se conecta el dispositivo final a un punto de alimentación hasta que se reciben todos los fragmentos del paquete Hello y se ha reensamblado el mensaje JSON.

  Como en el caso de estudio anterior, se hará uso del método *nanoTime()* de la clase System para realizar las mediciones.

  - ✓ Se definirá una variable de tipo *long* que almacene el tiempo de referencia: *long start.*
  - ✓ El primer tiempo se tomará en la clase principal de la aplicación *App* una vez se haya creado y mostrado la interfaz gráfica.

    static long start = System.nanoTime();

    El segundo valor se tomará en la clase subscriptora de este tipo de eventos y por lo tanto encargada de procesar los paquetes que contienen los fragmentos del mensaje JSON y reensamblarlos. La medida se hará justo después de haber sido reensamblado el mensaje JSON. Al igual que en el caso anterior la diferencia entre estos dos valores será mostrada por pantalla.

    System.out.println (System.nanoTime()- App.start));
  - ✓ Este proceso se repetirá 50 veces.

- Análisis de resultados: como en el caso anterior se mostrará una tabla con las muestras tomadas y seguidamente las gráficas que relacionan estos valores mediante la media, desviación estándar y la distribución. También se analizará cada uno de los casos y, por último, se extraerán ciertas conclusiones.

Caso 1. Distancia de dos metro sin obstáculos.

Muestras					
1	10,3688496	18	21,852832	35	20,3611778
2	5,23116121	19	20,6742774	36	3,80003548
3	4,77763149	20	21,1554424	37	3,97150548
4	20,5565393	21	20,8951409	38	3,81325825
5	20,5299809	22	4,01320077	39	20,1097019
6	5,26300375	23	4,23853464	40	3,87129413
7	4,68936805	24	20,1830477	41	4,20503751
8	20,5195754	25	4,09820708	42	20,0716633
9	20,7850636	26	20,1048965	43	20,6376104
10	20,5894543	27	20,5792424	44	20,0506075
11	19,4704649	28	20,2077573	45	20,1913133
12	4,86828259	29	19,8743291	46	4,02809278
13	20,498793	30	20,3766057	47	19,8249686
14	4,36449508	31	21,1795355	48	4,09552617
15	20,4501062	32	4,06039864	49	4,03723589
16	4,62006749	33	20,0813247	50	19,9807124
17	20,4735541	34	20,1814093		

Tabla 9. Muestras Caso 1 (Distancia dos metros sin obstáculos) en segundos.

A partir de las muestras tomadas se calculó el promedio y la desviación estándar:

Desvición	Media
7,87435729	14,0972463

Tabla 10. Desviación estándar y Media del Caso 1 (segundos).

En las siguientes gráficas se observa como las muestras se concentran en valores extremos, tiempos en torno a los 20 segundos y a los 5 segundos. Estas diferencias se deben a que a la hora de tomar estas medidas se tiene en cuenta el tiempo que tarda el dispositivo final en unirse a la red WSAN.

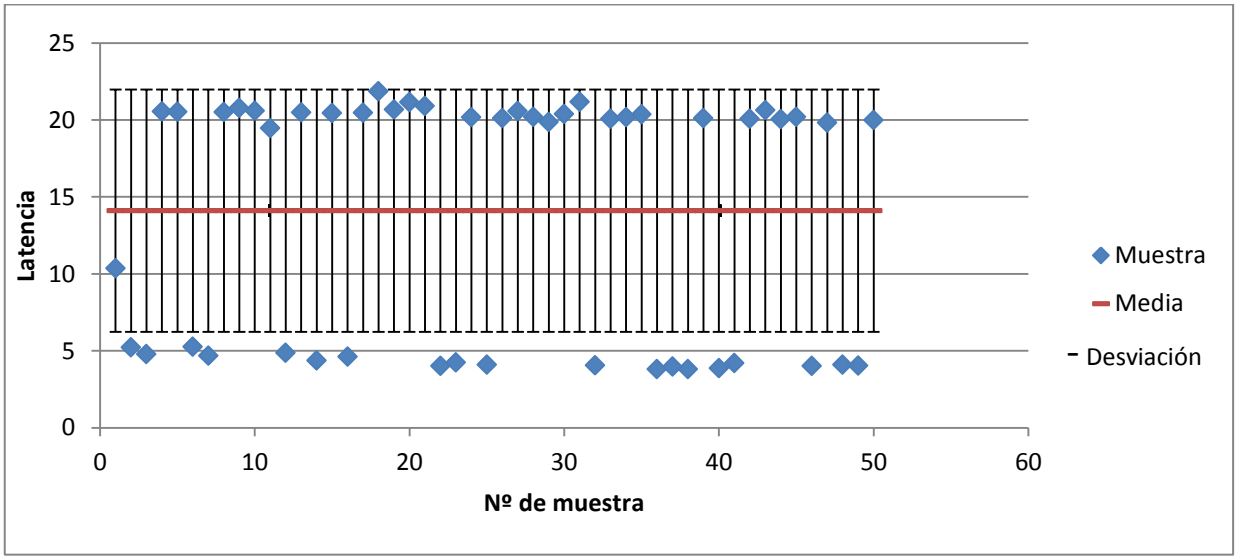


Figura 99. Gráfico de Línea con marcadores y media del Caso 1 en paquetes Hello (segundos).

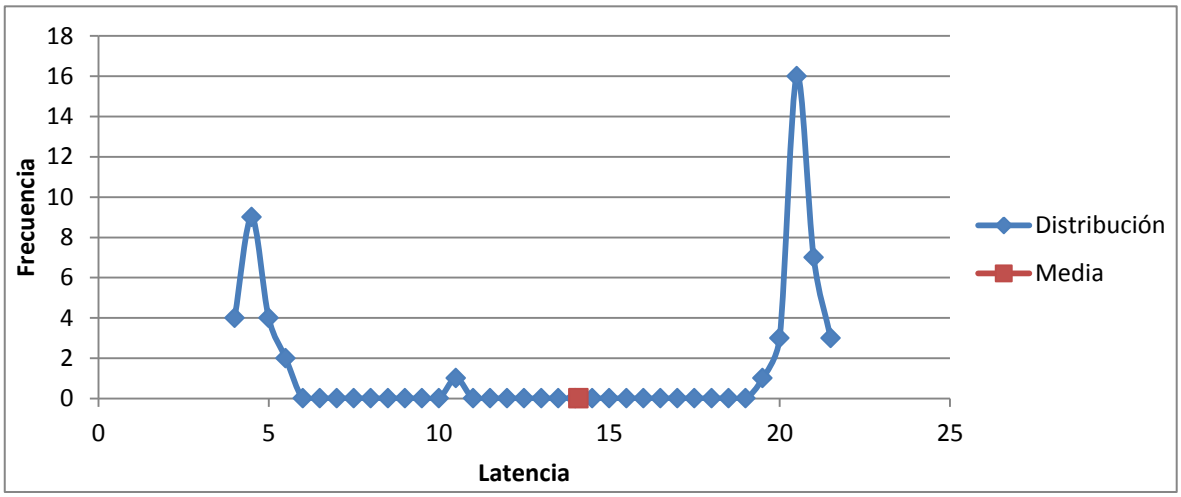


Tabla 11. Gráfico de Dispersión y media del Caso 1 (segundos).

- Conclusiones: teniendo en cuenta los resultados se extrae que el tiempo que transcurre en este caso es de unos 14 segundos de media. El principal motivo de una desviación típica tan alta es el tiempo que tardan los dispositivos en asociarse y conformar la red WSAN. También señalar que no se detectan cambios si se ponen obstáculos de por medio.

Capítulo 5:

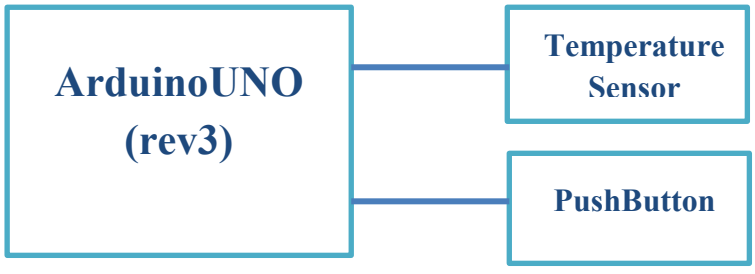
Futuros Trabajos

5.Futuros Trabajos

A pesar de los objetivos conseguidos en este proyecto, algunos aspectos interesantes han quedado fuera del alcance, pudiendo ser completados en futuros trabajos.

- Como se comentó en el apartado de conclusiones, la aplicación desarrollada en este proyecto permite la integración de nuevos dispositivos en el sistema, tanto placas ArduinoUNO (rev3) como nuevas plataformas hardware de distintos fabricantes. Sería interesante probar los resultados obtenidos en otras soluciones WSAN indicadas en el capítulo sobre el estudio del arte.
- Se puede plantear la ampliación del sistema para que deje de ser local y se convertir en remota, utilizando para ello servicios web, de manera que la aplicación esté alojada en un servidor y se ofrezca acceso remoto a los usuarios a través de una página web o aplicación distribuida.
- Una de las funcionalidades que quedó por desarrollar fue una comunicación dirigida por eventos. Esta posibilidad es útil para ciertas aplicaciones en las que se requiere un aviso por parte del dispositivo final, informando sobre algún cambio en el entorno.
- Se propone también la detección de desconexión de dispositivos finales. En general en las plataformas hardware viene integrado un botón que permite ser programado. Se busca la implementación de un mensaje de desconexión, en el que se indica al coordinador que ese dispositivo final se va a desconectar. El hardware utilizado en este proyecto no da soporte en este sentido pero a continuación se formula una posible solución:

  Desarrollar un sistema en el que la plataforma tenga conectado un botón que al pulsarlo antes de que se desconecte la plataforma envié un paquete de desconexión al coordinador indicando que se va a desconectar.



- Por otro lado, este proyecto no ha contemplado las funcionalidades de ahorro de batería, pero por ser un punto importante a tener en cuenta en una red

inalámbrica de sensores y actuadores se sugiere como posible estudio para futuros proyectos. Se propone estudiar el uso de otro tipo de alimentación como, baterías, placas solares, etc. en vez de mantener conectados los dispositivos a la red eléctrica. Habría que analizar el consumo energético por parte de los nodos, siendo la transmisión y recepción de datos por la interfaz radio el factor más determinante en éste.

GLOSARIO

Arduino	Plataforma open source para la creación de prototipos basada en software y hardware flexibles y fáciles de usar.
Eclipse IDE	Plataforma de desarrollo que incluye un conjunto de herramientas multiplataforma y de código abierto, soportando un gran número de lenguajes de programación.
IEEE 802.15.4	Es un estándar que define las capas física y de control de acceso al medio para redes inalámbricas de área personal con tasas bajas de transmisión de datos.
JSON	Es un formato de intercambio de datos ligero. Fácil de analizar y generar para las máquinas.
NetBeans IDE	Entorno de desarrollo integrado que permite a los programadores escribir, compilar, depurar y ejecutar programas principalmente escritos en Java. También lenguajes de programación como HTML5, PHP, C++, etc.
quanto	Es un periodo de tiempo que el sistema operativo le entrega a un proceso para que este utilice la CPU.
Red Ad-hoc	Red en la que no hay un nodo central, sino que todos los dispositivos están en igualdad de condiciones. Es el modo más sencillo para crear una red formada por un grupo de nodos móviles que forman una red temporal sin la ayuda de ninguna infraestructura externa.
Sketch	Nombre que reciben los programas desarrollados con el lenguaje Arduino. Se dividen en tres partes: estructura, valores y funciones.
SOA	Paradigma de arquitectura que permite diseñar y desarrollar sistemas software para proporcionar servicios para sistemas distribuidos en la red o para aplicaciones de usuario final.
Xbee	Módulos de radiofrecuencia fabricados por Digi International que poseen la capacidad de comunicarse con el microcontrolador a través de la comunicación UART y también tienen pines adicionales que pueden servir para aplicaciones adicionales.

ZigBee	Conjunto de protocolos de alto nivel que abarca los niveles de red y aplicación enfocados a las comunicaciones entre dispositivos de manera inalámbrica que requieren bajo consumo, alta seguridad y baja tasas de tráfico.

Anexo I:

Tabla Comparativa Motas

Anexo I

Durante el estudio de las plataformas hardware recientes para redes inalámbricas de sensores se realizó una tabla comparativa de dichas plataformas que permitiera el análisis de todas ellas en profundidad. Dicha tabla es demasiado grande como para poder exponerse con claridad, y se considera mucho más útil dividirla en bloques funcionales. Las tablas que se indican a continuación comparan las diferentes motas según los siguientes criterios: microcontrolador, radiofrecuencia, interfaces, sensores, software y otros.

	MICROCONTROLADOR			
NOMBRE DE LA MOTA	**MODELO**	**VELOCIDAD [MHz]**	**RAM [KB]**	**ROM [KB]**
CM3000	TI MSP430F1611	8	10	Program Flash Memory: 48
CM3300	TI MSP430F1611	8	10	Program Flash Memory: 48
CM4000	TI MSP430F1611	8	10	Program Flash Memory: 48
CM5000	TI MSP430F1611	8	10	Program Flash Memory: 48
CM5000 SMA	TI MSP430F1611	8	10	Program Flash Memory: 48
XM1000	TI MSP430F2618	16	8	Program Flash Memory: 116
Fleck3B	ATmega1281	8	4	Program Flash Memory: 128 EEPROM: 4
BTnode rev3	ATmega128L	7.37	64	Program Flash Memory: 128 EEPROM: 4
CoralMOTE	TI MSP430	8	10	Program Flash Memory: 48 EEPROM: 16
CoralMOTE IPv6	TI MSP430	8	10	Program Flash Memory: 48 EEPROM: 16

	MICROCONTROLADOR			
NOMBRE DE LA MOTA	MODELO	VELOCIDAD [MHz]	RAM [KB]	ROM [KB]
Waspmote v1.2 o WaspmotePRO	ATmega1281	14,7456	8	EEPROM: 4 (1kB reserved) FLASH: 128
SquidBee	ATmega168 (mayoría de las placas Arduino) ATmega8 (placas antiguas)	16	1	EEPROM: 512 bytes FLASH: 16
Lotus	ARM 7 Cortex M3 32-bit	10-100	64	Program Flash Memory: 512
IRIS	ATmega1281	8	8	Program Flash Memory: 128 EEPROM: 4
MICAz	ATmega128L	7.37	4	Program Flash Memory: 128 EEPROM: 4
MICA2	ATmega128L	7.37	4	Program Flash Memory: 128 EEPROM: 4
TelosB	TI MSP430	8	10	Program Flash Memory: 48 EEPROM: 16
TinyNode 584	TI MSP430	8	10	Program Flash Memory: 48 EEPROM: 16
Arduino UNO	ATmega328	16	2	Program Flash Memory: 32 EEPROM: 1
WSN-S-NK01 (multifuncional básico)	CC1110F32 (SoC)	13.5	4	Program Flash Memory: 32

	RADIOFRECUENCIA							
NOMBRE DE LA MOTA	TRANSCEPTOR	MODULACIÓN	BANDA [MHz]	VELOCIDAD DE DATOS [Kbps]	COBERTURA OUTDOOR [m]	COBERTURA INDOOR [m]	PROTOCOLO	SEGURIDAD
CM3000	CC2420 Antena externa omnidireccional con conector SMA.	O-QPSK	2400 - 2485	250	~300	40 - 50	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
CM3300	CC2420 Antena externa omnidireccional con conector SMA. Amplificador de potencia adicional & LNA (Low-Noise Amplifier)	O-QPSK	2400 - 2485	250	~800	80 - 120	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
CM4000	CC2420 Antena PCB direccional.	O-QPSK	2400 - 2485	250	~120	20 - 30	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
CM5000	CC2420 Antena PCB direccional.	O-QPSK	2400 - 2485	250	~120	20 - 30	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
CM5000 SMA	CC2420 Antena externa omnidireccional con conector SMA. Antena PCB direccional.	O-QPSK	2400 - 2485	250	~300	40 - 50	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
XM1000	CC2420 Antena PCB direccional.	O-QPSK	2400 - 2485	250	~120	20 - 30	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
Fleck3B	NRF905 (Nordic 905)	GFSK	433/868	250	1300		Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
BTnode rev3	CC1000 (ZV4002- Bluetooth)	FSK	868	76.8	> 100 con antena		No especificado	No especificado
CoralMOTE	CC2420	O-QPSK	2400 - 2485	250	~1000	~100	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
CoralMOTE IPv6	CC2420	O-QPSK	2400 - 2485	250	~1000	~100	Compatible con 802.15.4 y 6LoWPAN	La propia de 802.15.4 (AES-128)

	RADIOFRECUENCIA							
NOMBRE DE LA MOTA	TRANSCEPTOR	MODULACIÓN	BANDA [MHz]	VELOCIDAD DE DATOS [Kbps]	COBERTURA OUTDOOR [m]	COBERTURA INDOOR [m]	PROTOCOLO	SEGURIDAD
Waspmote v1.2 o WaspmotePRO	Algunos modelos Xbee Digi (802.15.4, ZigBee, RF 868/900 MHz) WiFi Module Libelium SIM900 SIMCom (GSM/GPRS) SIM5218E SIMCom (3G/GPS) Bluetooth Module Libelium 13.56 MHz RFID/NFC Module Libelium 125 kHz RFID Module Libelium	(depende del módulo RF)	868 900 850 1800 1900 2100 2400 (depende del módulo RF)	250 38.4 114 (depende del módulo RF)	500 - 12000 (depende del módulo RF)		802.15.4 ZigBee 868/900 MHz WiFi GSM/GPRS 3G/GPS Bluetooth RFID/NFC RFID	La propia de cada protocolo de comunicación utilizado
SquidBee	Xbee ZB XBee-PRO	QPSK	2400	250	>90 >1600 (Xbee Pro)	>30 >90 (Xbee Pro)	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
Lotus	Atmel RF231	O-QPSK	2405 - 2480	250	>500 >50 (on-board antenna)		802.15.4	La propia de 802.15.4 (AES-128)
IRIS	Atmel RF230	O-QPSK	2405 - 2480	250	> 300	>50	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
MICAz	CC2420	O-QPSK	2400 - 2483.5	250	75 - 100	20 - 30	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
MICA2	CC1000	FSK	868/916	38.4	152.4		Utilizan soluciones de comunicación no estandarizadas desarrolladas por la Universidad de Berkley.	No especificado
TelosB	CC2420	O-QPSK	2400 - 2483.5	250	75 - 100	20 - 30	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
TinyNode 584	XE1205	2 - FSK	868 - 870	1.2 - 152.3	200	40	No especificado	No especificado
Arduino UNO	XBee ZB XBee-PRO	QPSK	2400	250	120 1500 (Xbee Pro)	40 90 (Xbee Pro)	Compatible con 802.15.4	La propia de 802.15.4 (AES-128)
WSN-S-NK01 (nodo multifuncional básico)	CC1101	FSK OOK MSK GFSK	868	Hasta 500 (configurable).	290		Compatible con 802.15.4	La propia de 802.15.4 (AES-128)

	INTERFACES			
NOMBRE DE LA MOTA	SERIE	DIGITAL	ANALÓGICO	GATEWAY INFRASTRUCTURE
CM3000	SPI, UART, I2C	48 GPIO DAC	8 ADC	USB1000 de Advanticsys: USB para conectividad(USB2UART Rx/Tx) y reprogramación. SER1000 de Advanticsys: Tarjeta para conectividad (USB2UART Rx/Tx) y reprogramación.
CM3300	SPI, UART, I2C	48 GPIO DAC	8 ADC	USB1000 de Advanticsys: USB para conectividad(USB2UART Rx/Tx) y reprogramación. SER1000 de Advanticsys: Tarjeta para conectividad (USB2UART Rx/Tx) y reprogramación.
CM4000	SPI, UART, I2C	48 GPIO DAC	8 ADC	USB1000 de Advanticsys: USB para conectividad(USB2UART Rx/Tx) y reprogramación. SER1000 de Advanticsys: Tarjeta para conectividad (USB2UART Rx/Tx) y reprogramación.
CM5000	SPI, UART, I2C, USB	48 GPIO DAC	8 ADC	SER1000 de Advanticsys: Tarjeta para conectividad (USB2UART Rx/Tx) y reprogramación.
CM5000 SMA	SPI, UART, I2C, USB	48 GPIO DAC	8 ADC	SER1000 de Advanticsys: Tarjeta para conectividad (USB2UART Rx/Tx) y reprogramación.
XM1000	SPI, UART, I2C, USB	48 GPIO	8 ADC	No especificado
Fleck3B	SPI, 2 UART, I2C, RS-232	25 GPIO	7 ADC	No especificado
BTnode rev3	SPI, UART,I2C, PWM	GPIO	ADC	No especificado
CoralMOTE	SPI, UART,I2C	GPIO	ADC	Gateway de Inetsis
CoralMOTE IPv6	SPI, UART,I2C	GPIO	ADC	Gateway de Inetsis

	INTERFACES			
NOMBRE DE LA MOTA	SERIE	DIGITAL	ANALÓGICO	GATEWAY INFRASTRUCTURE
Waspmote v1.2 o WaspmotePRO	I2C, USB, SPI, PWM, 2X UART, Sockets específicos para los "sensores básicos": temperatura, humedad y luz.	8 DIO	7 ADC	Waspmote Gateway (Comunicaciones: USB PC - 802.15.4/ZigBee) Waspmote GPRS/GSM Meshlium (Comunicaciones Wifi (2.4 GHz-5GHz), ZigBee, 3G/GPRS, Bluetooth y Ethernet).
SquidBee	PWM, USB, SPI, I2C, UART	12 DIO	6 ADC	USB integrado en la placa para programación, comunicación. SquidBee Gateway. Compatible con Meshlium.
Lotus	I2C, I2S, SPI, 3X UART, USB	GPIO	ADC	No especificado
IRIS	I2C, SPI, UART	DIO	ADC	USB/serie para un ordenador local o Ethernet para una estación base.
MICAz	I2C, SPI, 2 UART	DIO	8 ADC	USB/serie para un ordenador local o Ethernet para una estación base.
MICA2	I2C, SPI, 2 UART	DIO	8 ADC	USB/serie para un ordenador local o Ethernet para una estación base.
TelosB	I2C, SPI, UART, USB	DAC	ADC	USB integrado en la placa para programación, comunicación y recogida de datos
TinyNode 584	UART, SPI, LVTTL	19 DIO	8 ADC	Puerto serie a adaptador USB
Arduino UNO	I2C, SPI, 6 PWM, UART, USB, LED, reset button	14 DIO	6 ADC	USB integrado en la placa para programación, comunicación.
WSN-S-NK01 (nodo multifuncional básico)	I2S, SPI, UART, USB	21 GPIO	8 ADC	WSN-SUM-USB

	SENSORES	
NOMBRE DE LA MOTA	PLACAS DE SENSORES	SENSORES (incluidos en la propia placa)
CM3000	Conector Hirose de 51 pines permite incorporar cualquiera de las tarjetas sensoras del catálogo de Advanticsys: temperatura, humedad, iluminancia, acelerómetro, CO & CO2, PIR, inclinómetro, etc.	Ninguno
CM3300	Conector Hirose de 51 pines permite incorporar cualquiera de las tarjetas sensoras del catálogo de Advanticsys: temperatura, humedad, iluminancia, acelerómetro, CO & CO2, PIR, inclinómetro, etc.	Ninguno
CM4000	Conector Hirose de 51 pines permite incorporar cualquiera de las tarjetas sensoras del catálogo de Advanticsys: temperatura, humedad, iluminancia, acelerómetro, CO & CO2, PIR, inclinómetro, etc.	Ninguno
CM5000	No soporta tarjetas de sensores de Advanticsys	Temperatura Humedad relativa Luz visible e infrarroja
CM5000 SMA	No soporta tarjetas de sensores de Advanticsys	Temperatura Humedad relativa Luz visible e infrarroja
XM1000	No soporta tarjetas de sensores de Advanticsys	Temperatura Humedad relativa Luz visible e infrarroja
Fleck3B	Tarjetas de sensores para la serie Fleck Sensores ambientales: aire, tierra, agua. Posicionamiento: GPS, 6DoF IMU (inertial measurement unit): giroscopio, acelerómetros y magnetómetros. Procesamiento de señal (DSP con interfaz audio y placas para cámara).	Temperatura Luminosidad
BTnode rev3	Extensiones para placa de sensores genéricos (J1 Hirose Receptacle DF17-40DS, J2 Molex 53261-1590)	Ninguno
CoralMOTE	Pueden añadirse otros sensores, en función de las necesidades de la red: CO, CO2, caudal, etc. Pueden conectarse a diferentes actuadores para el control automático de sistemas de iluminación y climatización o de cualquier otro dispositivo.	(Disponible con o sin sensoses) Temperatura, Humedad Luminosidad, Presencia (PIR) *Actuadores integrados: Switching Relé
CoralMOTE IPv6	Pueden añadirse otros sensores, en función de las necesidades de la red: CO, CO2, caudal, etc. Pueden conectarse a diferentes actuadores para el control automático de sistemas de iluminación y climatización o de cualquier otro dispositivo.	(Disponible con o sin sensoses) Temperatura, Humedad, Luminosidad, Presencia (PIR) *Actuadores integrados: Switching Relé

	SENSORES	
NOMBRE DE LA MOTA	PLACAS DE SENSORES	SENSORES (incluidos en la propia placa)
Waspmote v1.2 o WaspmotePRO	Complatible con placas de sensores v2.0 de Libelium. No plenamente compatibles con placas de sensores v1.0 de Libelium. Gases, eventos, smart cities, smart parking, agricultura, videocámara, radiación, smart metering.	Temperatura Acelerómetro
SquidBee	Se pueden conectar hasta 18 sensores(sondas de longitud variable): sonido, gases, humo, presencia, GPS, presión	Temperatura Humedad Luminosidad
Lotus	Conector de expansión para la Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.	Ninguno
IRIS	Conector de expansión para la Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.	Ninguno
MICAz	Conector de expansión para la Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.	Ninguno
MICA2	Conector de expansión para la Luz, Temperatura, RH, Presión barométrica, Aceleración/Sismica, Acústica, Magnética y otras placas de sensores MEMSIC.	Ninguno
TelosB	No incorpora ningún conector de expansión para sensores.	Temperatura Luz Humedad
TinyNode 584	Posibilidad de ampliar con nuevos sensores y actuadores a través de Standard Extension Board (que lleva integrados un sensor de luz, temperatura y humedad) de TinyNode.	Temperatura
Arduino UNO	Posibilidad de conectar sensores y actuadores a través de los pines analógicos y digitales.	Ninguno
WSN-S-NK01 (nodo multifuncional básico)	Conector E/S de expansión que permite la interconexión de otros sensores / actuadores.	Temperatura Pulsador

	SOFTWARE			
NOMBRE DE LA MOTA	SISTEMA OPERATIVO	PLATAFORMA DE DESARROLLO	LENGUAJE DE PROGRAMACIÓN	LICENCIA
CM3000	TinyOS 2.x ContikiOS	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
CM3300	TinyOS 2.x ContikiOS	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
CM4000	TinyOS 2.x ContikiOS	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
CM5000	TinyOS 2.x ContikiOS	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
CM5000 SMA	TinyOS 2.x ContikiOS	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
XM1000	TinyOS 2.x ContikiOS	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
Fleck3B	TinyOS FOS	Eclipse y plugin NESCDT	nesC	Open-source, BSD-licensed operating system
BTnode rev3	Nut/OS TinyOS	BTNut System Software Eclipse y plugin NESCDT	nesC C	Open-source, BSD-licensed operating system (TinyOS)
CoralMOTE	TinyOS Contiki	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system
CoralMOTE IPv6	TinyOS Contiki	Eclipse y plugin NESCDT Instant Contiki	nesC C	Open-source, BSD-licensed operating system

	SOFTWARE			
NOMBRE DE LA MOTA	**SISTEMA OPERATIVO**	**PLATAFORMA DE DESARROLLO**	**LENGUAJE DE PROGRAMACIÓN**	**LICENCIA**
Waspmote v1.2 o WaspmotePRO	No se ejecuta ningún sistema operativo en el microcontrolador, solo un gestor de arranque para la carga de aplicaciones por USB.	Waspmote IDE	C++	GNU
SquidBee	DuinOS ArdOS	Arduino IDE	Lenguaje Arduino(basado en C/C++ con algunas modificaciones leves)	Open Hardware y open source
Lotus	Soporte por defecto de RTOS TinyOS FreeRTOS	IAR Embedded Workbench Eclipse y plugin NESCDT Mote Runner	C/C++ (IAR Systems) nesC (TinyOS) Java/C# (MoteRunner)	Open-source, BSD-licensed operating system
IRIS	TinyOS	Mote Runner Eclipse y plugin NESCDT MoteWorks	Java/C# (MoteRunner) nesC (TinyOS, MoteWorks)	Open-source, BSD-licensed operating system
MICAz	TinyOS, SOS, NanoRK	MoteWorks Eclipse y plugin NESCDT	nesC (TinyOS, MoteWorks)	Open-source, BSD-licensed operating system
MICA2	TinyOS, SOS, NanoRK	MoteWorks Eclipse y plugin NESCDT	nesC (TinyOS, MoteWorks)	Open-source, BSD-licensed operating system
TelosB	TinyOS 1.1.11 o superior	Eclipse y plugin NESCDT	nesC (TinyOS)	Open-source, BSD-licensed operating system
TinyNode 584	TinyOS	Eclipse y plugin NESCDT	nesC (TinyOS)	Open-source, BSD-licensed operating system
Arduino UNO	DuinOS ArdOS	Arduino IDE	Lenguaje Arduino(basado en C/C++ con algunas modificaciones leves)	Open Hardware y open source
WSN-S-NK01 (nodo multifuncional básico)	FreeRTOS	IAR Embedded Workbench Code Composer Essentials	C/C++ (IAR Systems)	Open-source

	OTROS			
NOMBRE DE LA MOTA	TIPO DE ALIMENTACIÓN	CONSUMO	PRECIO	EXTRAS
CM3000	2 pilas tipo AA Alimentación externa: USB1000	Parado: 2.1 µA Tx/Rx: 17.33/19.13 [mA]	71 €	Memoria flash para almacenar medidas: 1024 KB
CM3300	2 pilas tipo AA Alimentación externa: USB1000, DC Power Input	Parado: 5 µA Tx/Rx: 120.33/30.33 [mA]	95 €	Memoria flash para almacenar medidas: 1024 KB
CM4000	2 pilas tipo AA Alimentación externa: USB1000	Parado: 2.1 µA Tx/Rx: 17.73/19.13 [mA]	71 €	Memoria flash para almacenar medidas: 1024 KB
CM5000	2 pilas tipo AA Alimentación externa: USB	Parado: 2.1 µA Tx/Rx: 17.73/19.13 [mA]	77 €	Memoria flash para almacenar medidas: 1024 KB
CM5000 SMA	2 pilas tipo AA Alimentación externa: USB	Parado: 2.1 µA Tx/Rx: 17.73/19.13 [mA]	85 €	Memoria flash para almacenar medidas: 1024 KB
XM1000	2 pilas tipo AA Alimentación externa: USB	Parado: 2.1 µA Tx/Rx: 17.73/19.13 [mA]	85 €	Memoria flash para almacenar medidas: 1024 KB
Fleck3B	2 pilas tipo AA recargables Circuitos integrados que permiten conectar una celula solar	Parado: 10.5 µA Tx/Rx: 38/20.5[mA]	No especificado	Memoria flash para almacenar medidas: 1024 KB Conector SMA para antena externa
BTnode rev3	2 pilas tipo AA Alimentación externa: 3.8 V – 5 V	Parado: 9.9 mW Tx/Rx: 105.6 [mW]	205 €	Radio Bluetooth 1.2 Zeevo y plugin BTSense
CoralMOTE	2 pilas tipo AA Alimentación externa: panel solar, conexión a la red eléctrica (220V AC).	Parado: 6.1 µA Tx/Rx: 19.2/20.6 [mA]	140€ (Sin sensores) 180€ (Con sensores: Temperatura, humedad relativa y luminosidad) 190€ (Con sensor: PIR) 228€ (Con relé inalambrico para encender y apagar hasta 8 dispositivos)	Memoria flash para almacenar medidas: 1024 KB. Conector SMA para antena externa (Hembra 50 ohm). Encapsulado con botones y leds de estados. Coral2K: solución que permite monitorizar y controlar entornos mediante una WSAN.
CoralMOTE IPv6	2 pilas tipo AA Alimentación externa: panel solar, conexión a la red eléctrica (220V AC).	Parado: 6.1 µA Tx/Rx: 19.2/20.6 [mA]	No especificado	Mismos extras que CoralMOTE

	OTROS			
NOMBRE DE LA MOTA	**TIPO DE ALIMENTACIÓN**	**CONSUMO**	**PRECIO**	**EXTRAS**
Waspmote v1.2 o Waspmote PRO	2 pilas tipo AA Alimentación externa: USB-220V, Panel solar rigido 7V-500mA, Panel solar flexible 7.2V-100mA.	Parado: 55 µA Tx/Rx: 15 [mA]	Desde 130€ hasta 233€ (dependiendo del módelo RF, placa radio de expansión, placas de sensores)	Tarjeta SD: 2 GB Soporta programación OTA Placa radio de expansión (permite conectar dos radios al mismo tiempo). Hibernate mode: 0.06µA.
SquidBee	Alimentación externa: USB 4 pilas tipo AA o 2 pilas 6F22	Depende del módulo de RF	150 €	Módulo GPRS de Libelium compatible
Lotus	2 pilas tipo AA Alimentación externa: 2.7 V - 3.3 V	Parado: 10 µA	WSN-START2110CB 865 € (2 nodos con tarjetas de sensores MTS400 y estación base encapsulada) WSN-PRO2110CB 2135 € (6 nodos son sensores, 1 estación base encapsulada, 1 tarjeta de programación MIB520, 1 tarjeta de adquisición MDA300, 1 transmisor XM2110 para tarjeta adquisición)	Measurement Serial Flash: 64MB USB Client con conector mini-B en la placa
IRIS	2 pilas tipo AA Alimentación externa: 2.7 V - 3.3 V	Parado: 8 µA Tx/Rx: 25/24 [mA]	WSN-START2110CB 865 € (2 nodos con tarjetas de sensores MTS400 y estación base encapsulada) WSN-PRO2110CB 2135 € (6 nodos son sensores, 1 estación base encapsulada, 1 tarjeta de programación MIB520, 1 tarjeta de adquisición MDA300, 1 transmisor XM2110 para tarjeta adquisición)	Memoria flash para almacenar medidas: 512 KB
MICAz	2 pilas tipo AA Alimentación externa: 2.7 V - 3.3 V	Parado: 16 µA Tx/Rx: 25.4/27.7 [mA]	WSN-START2110CB 865 € (2 nodos con tarjetas de sensores MTS400 y estación base encapsulada) WSN-PRO2110CB 2135 € (6 nodos son sensores, 1 estación base encapsulada, 1 tarjeta de programación MIB520, 1 tarjeta de adquisición MDA300, 1 transmisor XM2110 para tarjeta adquisición)	Memoria flash para almacenar medidas: 512 KB

	OTROS			
NOMBRE DE LA MOTA	**TIPO DE ALIMENTACIÓN**	**CONSUMO**	**PRECIO**	**EXTRAS**
MICA2	2 pilas tipo AA Alimentación externa: 2.7 V - 3.3 V	Parado: 16 µA Tx/Rx: 35/18 [mA]	WSN-START2110CB 865 € (2 nodos con tarjetas de sensores MTS400 y estación base encapsulada) WSN-PRO2110CB 2135 € (6 nodos son sensores, 1 estación base encapsulada, 1 tarjeta de programación MIB520, 1 tarjeta de adquisición MDA300, 1 transmisor XM2110 para tarjeta adquisición)	Memoria flash para almacenar medidas: 512 KB
TelosB	2 pilas tipo AA Alimentación externa: USB	Parado: 6.1 µA Tx/Rx:19.2/20.6 [mA]	WSN-START2110CB 865 € (2 nodos con tarjetas de sensores MTS400 y estación base encapsulada) WSN-PRO2110CB 2135 € (6 nodos son sensores, 1 estación base encapsulada, 1 tarjeta de programación MIB520, 1 tarjeta de adquisición MDA300, 1 transmisor XM2110 para tarjeta adquisición)	Memoria flash para almacenar medidas: 1024 KB
TinyNode 584	2 pilas AA alcalinas o 1 pila de litio.	Parado: 7 µA Tx/Rx: 62/16 [mA]	109 €	Memoria flash para almacenar medidas: 512 Conectores SMA o MMCX.
Arduino UNO	Alimentación externa: USB, adaptador AD-DC (conector de alimentación de la placa), batería (en los pines GND y Vin)	Depende del módulo RF	27 €	
WSN-S-NK01 (multifuncional básico)	USB Alimentación externa: pilas, conexión a la red eléctrica (220V AC).	Parado: 0.3 µA Tx/Rx: 16/18 [mA]	MóduloWSN-S-NK01: 60 € Dubugger: 80 € Kit de desarrollo:550 € (1 sniffer, 1 recolector USB, 1 debugger, 4 nodos multifuncionales básicos, CDs software y documentación)	Debugger: dispositivo imprescindible para la programación y depurado de aplicaciones (conexión USB).

Bibliografía

[1] TinyOS. Página oficial del Sistema Operativo TinyOS. [Online]. http://www.tinyos.net/

[2] Contiki. Página oficial del Sistema Operativo ContikiOS. [Online]. http://www.contiki-os.org/

[3] FreeRTOS. Página oficial del Sistema Operativo FreeRTOS. [Online]. http://www.freertos.org

[4] ETH Zurich. BTnut System Software. [Online]. www.btnode.ethz.ch/static_docs/doxygen/btnut/

[5] Peter Corke and Pavan Sikka, "FOS-a new operating system for sensor networks,".

[6] DuiOS. Página oficial del Sistema Operativo DuinOS. [Online]. https://github.com/DuinOS

[7] Welcome to ArdOS - The Arduino Operating System. [Online]. https://bitbucket.org/ctank/ardos-ide/wiki/Home

[8] ArdOS Overview. [Online]. https://bitbucket.org/ctank/ardos/overview

[9] SOS. Página oficial del Sistema Operativo SOS. [Online]. https://projects.nesl.ucla.edu/public/sos-2x/doc/

[10] Nano-RK. A Wireless Sensor Networking Real-Time Operating System. [Online]. http://www.nanork.org/projects/nanork/wiki

[11] IBM. Página oficial IBM. [Online]. http://www.zurich.ibm.com/moterunner/

[12] T Kramp et al., "IBM Mote Runner-Executive Summary," 2008.

[13] "MoteWorks Getting Started Guide," 2012.

[14] Texas Instruments. Code Composer Studio (CCStudio) Integrated Development Environment (IDE) v5. [Online]. http://www.ti.com/tool/ccstudio

[15] The Eclipse Foundation. Página oficial de Eclipse Foundation. [Online]. http://www.eclipse.org

[16] TinyOS. NESCDT-An editor for nesC in Eclipse. [Online]. http://docs.tinyos.net/tinywiki/index.php/NESCDT-_An_editor_for_nesC_in_Eclipse#Extending_and_Customizing_the_Plugin

[17] Contiki. Página oficial de Instant Contiki. [Online]. http://www.contiki-os.org/start.html

[18] "Waspmote Quickstart Guide," 2013.

[19] Arduino. Página oficial de Arduino- Arduino Development Environment. [Online].

http://arduino.cc/en/Guide/Environment

[20] David Gay, Philip Levis, David Culler, and Eric Brewer, "nesC Language Reference Manual,".

[21] Arduino. Página oficial de Arduino. [Online]. http://www.arduino.cc

[22] Michael Margolis, *Arduino Cookbook*.: O´REILLY.

[23] Geir E. Øien, ""“ZigBee and IEEE 802.15.4: A brief introduction”.,".

[24] Muñoz Castejón Rodrigo, ""’Interconexión de redes de sensores inalámbricos 802.15.4 en localizaciones remotas”,".

[25] ZigBee Alliance. Página oficial de ZigBee. [Online]. www.zigbee.org

[26] Valverde Rebaza Jorge Carlos, ""’El estándar inalámbrico Zigbee”,".

[27] DASH7 Alliance. DASH7 ALLIANCE. [Online]. http://www.dash7.org

[28] Haystack. Haystack Technologies. [Online]. http://haystacktechnologies.com

[29] JP Norair Co-chair, "Introduction to DASH7 Technology," June 24, 2009. [Online]. http://www.slideshare.net/jpnorair/introduction-to-dash7-technology-1698125

[30] 6LoWPAN. Página Oficial de 6LoWPAN. [Online]. http://www.6lowpan.org/#

[31] Bluetooth. Página oficial de Bluetooth. [Online]. www.bluetooth.org

[32] Libelium. Página oficial de Libelium. [Online]. http://www.libelium.com/

[33] Manu Arenas. (2013, Junio) Entrevista con Javier Martínez, Sales Manager de Libelium. [Online]. http://www.smartscities.com/index.php/component/k2/item/329-entrevista-con-javier-martinez-sales-manager-de-libelium

[34] Libelium, Spin-off de la UZ, pionera en tecnología de redes sensoriales. [Online]. http://www.aragoninvestiga.org/libelium-spin-off-de-la-uz-pionera-en-tecnologia-de-redes-sensoriales/

[35] Libelium, "Datasheet SquidBee,".

[36] Presentación de Waspmote: nueva plataforma para Redes Sensoriales Inalámbricas. [Online]. http://www.aragoninvestiga.org/Presentacion-de-Waspmote-nueva-plataforma-para-Redes-Sensoriales-Inalambricas/

[37] Libelium. Waspmote Datasheet. [Online]. http://www.libelium.com/development/waspmote/documentation/waspmote-datasheet/

[38] Libelium. Catálogo Wireless Sensor Networks 2013. [Online]. http://www.libelium.com/xhjs76gd/libelium_products_catalogue.pdf

[39] Jason Lester Hill, "System Architecture for Wireless Sensor Networks," 2003.

[40] Crossbow. Página oficial de Crossbow. [Online]. http://www.xbow.com/

[41] MEMSIC. Página oficial de MEMSIC. [Online]. http://www.memsic.com

[42] MEMSIC. MEMSIC Completes Crossbow Technology Acquisition. [Online]. http://www.memsic.com/about-memsic/news-room.cfm?nid=MEMSIC%20Completes%20Crossbow%20Technology%20Acquisition

[43] MEMSIC. (2012, April) MEMSIC Introduces LOTUS - Next-Generation Mote Platform for High-Performance Wireless Sensor Networks. [Online]. http://investor.memsic.com/releasedetail.cfm?ReleaseID=573037

[44] MEMSIC. Datasheet de LOTUS. [Online]. http://www.memsic.com/wireless-sensor-networks/

[45] (2007, April) Crossbow Introduces IRIS Wireless Product Line. [Online]. http://www.sensorsmag.com/wireless-applications/news/crossbow-introduces-iris-wireless-product-line-2487

[46] IBM and MEMSIC Partner on Wireless Sensors. [Online]. http://www.sensorsmag.com/networking-communications/wireless/news/ibm-and-memsic-partner-wireless-sensors-7208

[47] David Culler, U.C. Berkeley, Jason Hill, U.C. Berkeley, Mike Horton, Crossbow Technology, Inc. , Kris Pister, U.C. Berkeley, Robert Szewczyk, U.C. Berkeley, Alec Woo, U.C. Berkeley. (2002, April) MICA: The Commercialization of Microsensor Motes. [Online]. http://www.sensorsmag.com/networking-communications/mica-the-commercialization-microsensor-motes-1070

[48] MEMSIC. Wireless Modules. [Online]. http://www.memsic.com.php5-12.dfw1-1.websitetestlink.com/products/wireless-sensor-networks/wireless-modules.html

[49] MEMSIC. Datasheet MICAz. [Online]. http://www.memsic.com/wireless-sensor-networks/

[50] Sentilla. Página oficial de Sentilla. [Online]. http://www.sentilla.com/

[51] Sentilla. Documents Back by Popular Demand. [Online]. http://sentilla.tomcoh.com/blogs/2007/documents-back-popular-demand

[52] Arduino. ArduinoBoard UNO. [Online]. http://arduino.cc/en/Main/ArduinoBoardUno

[53] Arduino. Arduino Forum. [Online]. http://www.forum.arduino.cc

[54] Why the arduino won and why its here to stay. [Online]. http://makezine.com/2011/02/10/why-the-arduino-won-and-why-its-here-to-stay/

[55] Arduino. Arduino Uno-Communication. [Online]. http://arduino.cc/en/Main/ArduinoBoardUno#.UxMGV_l5P00

[56] Arduino. XBee Shield. [Online]. http://arduino.cc/en/Main/ArduinoXbeeShield

[57] Arduino. Wireless SD Shield. [Online].

http://arduino.cc/en/Main/ArduinoWirelessShield

[58] Arduino. Wireless Shield SD with XBee Series 2 radios. [Online]. http://arduino.cc/en/Guide/ArduinoWirelessShieldS2

[59] Andrés Duarte. XBee y Arduino. [Online]. http://www.artinteractivo.com/xbee-y-arduino

[60] Robert Faludi, *Building Wireless Sensor Networks*.: O´Reilly, 2010.

[61] SparkFun Electronics. XBee Explorer USB. [Online]. https://www.sparkfun.com/products/8687

[62] Adafruit Industries. XBee Adapter. [Online]. http://www.adafruit.com/products/126

[63] Digi International, Inc. XBee products. [Online]. http://www.digi.com/products/xbee/

[64] Digi International, Inc., "XBee ZigBee product manual," 2013.

[65] Digi International, Inc., "X-CTU Configuration & Test Utility Software,".

[66] Digi Inc. X-CTU Software. [Online]. http://www.digi.com/support/productdetail?pid=3352

[67] Digi Inc. Configuring XBee Radios with X-CTU. [Online]. http://examples.digi.com/get-started/configuring-xbee-radios-with-x-ctu/

[68] XBee Sleeping. [Online]. http://citizen-sensing.org/2013/08/xbee-sleeping/

[69] Arduino. SoftwareSerial Library. [Online]. http://arduino.cc/en/Reference/SoftwareSerial#.UxGoKPl5PQi

[70] Mikal Hart. Arduiniana. [Online]. http://arduiniana.org/libraries/newsoftserial/

[71] xbee-api. A Java API for Digi XBee/XBee-Pro OEM RF Modules. [Online]. https://code.google.com/p/xbee-api/wiki/WhyApiMode

[72] Ismail H. Kasimoglu Ian F. Akyldiz, "Wireless sensor and actor networks: research challenges," 2004.

[73] Bahareh Gholamzadeh and Hooman Nabovati, "Concepts for Designing Low Power Wireless Sensor Network," 2008.

[74] Ana-Belén García-Hernando, José-Fernán Martínez-Ortega, Juan-Manuel López-Navarro, Aggeliki Prayati, and Luis Redondo-López, "Problem Solving for Wireless Sensor Networks," 2008.

[75] Fei Hu: Xiaojun Cao, "Wireless Sensor Networks: Principles and Practice," 2012.

[76] I.F. Akyldiz, W. Su, and E. Cayirci Y. Sankarasubramaniam, "Wireless sensor networks: a survey," 2002.

[77] MIT (Massachusetts Institute of Technology), "10 Emerging Technologies That Will Change the World," 2003.